A flexible model supporting QoS and reallocation for grid applications

Al-Bodour, Reda

Award date:
2011

Awarding institution:
Coventry University

A Flexible Model Supporting QoS and Reallocation for Grid Applications

Reda Al Bodour

2011

Abstract

The rise of business-oriented and commercial applications for Grid computing environments has recently gathered pace. Grid computing traditionally has been linked with scientific environments, where heterogeneous resources provided by Grid systems and infrastructures were employed for carrying out computationally-intensive and data-intensive scientific experiments or applications that may have not been possible before. The natural progression is that business-oriented applications will look to build on this success and utilise the large number of heterogeneous Grid resources including computational resources such as CPUs and memory and storage resources such as disk space, potentially available. The success of introducing these applications into the mainstream is directly related to whether service providers can deliver a level of Quality of Service (QoS) to a consumer and the ability of the consumer to request high-level QoS such as the numbers of CPUs required or the RAM required.

QoS refers to the guidelines and requirements requested by a user/consumer from the service providers and resources. The communication and agreement establishment processes between user and provider must be defined clearly to accommodate a new type of user where knowledge of the underlying infrastructure cannot be assumed. QoS parameters have generally been defined at the Grid resource level using low level definitions. This tailors to specific applications and models related to scientific domains where brokering, scheduling and QoS delivery is designed for specific applications within specific domains.

This thesis presents a flexible model for high-level QoS requests. Business Grid Quality of Service (BGQoS) is introduced for business-oriented and commercial Grid applications which may wish to make use of the resources made available by Grid system environments. BGQoS allows *GRCs (Grid Resource Consumers)* to specify varying types of high-level QoS requirements which are delivered via querying up-to-date resource information, matchmaking and monitoring operations. Moreover, we present dynamically calculated metrics for measuring QoS such as reliability, increasing the accuracy of meeting the GRC's requirements. On the other hand *GRPs (Grid Resource Provider)* are also capable of advertising their resources, their capabilities, their usage policies and availability both locally and globally. This leads to a flexible model that could be carried across domains without altering the core operations and which could easily be expanded in order to accommodate different types of GRC, resources and applications.

Acknowledgements

I would like to express my deepest appreciation and gratitude to my Director of Studies Professor Anne James, for her guidance, advice, experience and patience. I would like to thank her giving me this opportunity. I would also like to thank my supervisor Dr. Norlaily Yaacob for her encouragement, guidance and attention to detail. Both have provided me with the attention and effort that has helped me throughout my time as a PhD student. I would also like to express my appreciation for Dr. Anthony Godwin for his input, advice and effort.

Thank you to all the staff at Coventry University who have tirelessly helped. Thank you to my colleagues, fellow students and fellow members of my research group. You have been kind and friendly throughout the years and for that I am thankful.

My father, Prof. Salman Albdour: Thank you for being there when I needed you, for guiding me through the years, for your support, for your advice, for your talks and for listening. Thank you for everything. Without you none of this would be possible.

My mother, Fereshteh Pourreza: Thank you for your support, for your patience, for your kindness when I needed it and for your encouragement when I needed it. Thank you for keeping my feet on the ground and thank you for your love and compassion.

My brother, Mohammad Al-Bodour: You have had to put up with the day to day stress that accompanies a PhD and you have done so with patience and love. I am grateful that you are in my life. Thank you.

Finally, thank you to all my friends. You have been amazing.

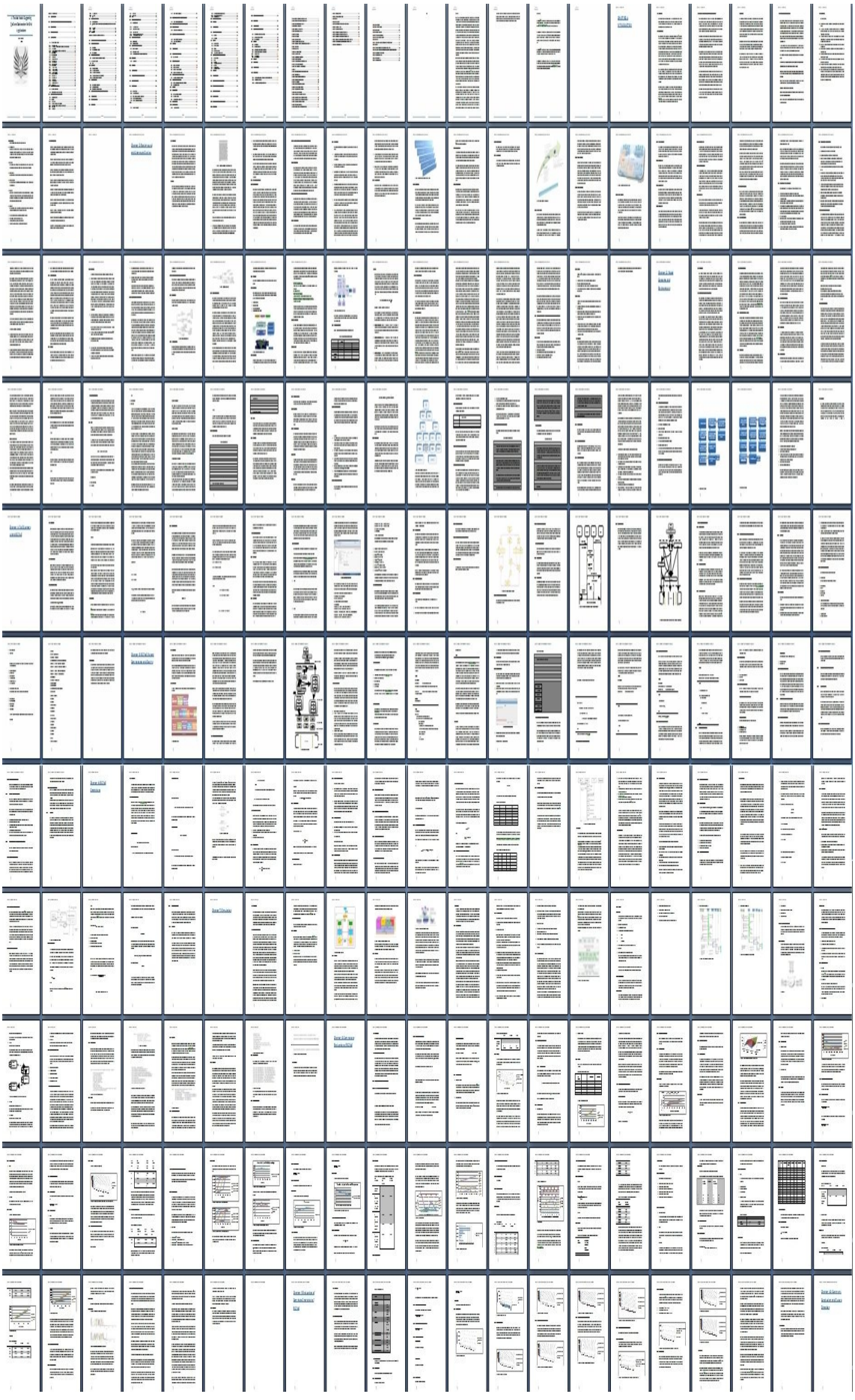
To my parents – Everything I was, am and will be, I owe to you.

Contents

Figures

Tables

2011



2011

Publications

Albodour, R. and James, A. and Yaacob, N. and Godwin, A. (2008) *Grid service QoS in medical environments*. 'The 12th International Conference on Cooperative Work in Design', held 16-18 April in Xi'an, China. ISBN: 978-1-4244-1650-9. 548 – 552

Albodour, R. and James, A. and Yaacob, N. and Godwin, A. (2008) 'QoS Requirements for a Medical Application on the Grid'. *Computer Supported Cooperative Work in Design IV* Lecture Notes in Computer Science, 2008, Volume 5236/2008, 316-330, DOI: 10.1007/978-3-540-92719-8_29 : Springer-Verlag, Berlin, Heidelberg, 316-330

Albodour, R. and James, A. and Yaacob, N. (2010) *An extension of GridSim for quality of service*. 'The 14th International Conference on Cooperative Work in Design', held 14-16 April in Shanghai, China. ISBN: 978-1-4244-6763-1, 361 - 366

Albodour, R. and James, A. and Yaacob, N. (2011)' High Level QoS-Driven Model for Grid Applications in a Simulated Environment', *Future Generation Computer Systems, The International Journal of Grid Computing and eScience Special Section "Quality of Service in Grid and Cloud Computing" (in press)*

2011

CHAPTER 1:

INTRODUCTION

1.1. Introduction

This chapter serves three main purposes. First, it introduces the general setting and motivation behind the research. Second, it specifies the research objectives, including the research question and the contributions. Finally, it provides an overview of the methodology used and the structure of this thesis.

1.2. General Setting

The emergence of Grid Computing as a mainstream solution (Scale Out Software 2011) has allowed the progression and development of a new generation of applications that utilise the resources Grids provide. Grids are systems that provide the user with seamless access to a variety of resources, such as CPUs, storage space, data and instruments. The Grid computing field is the result that has emerged from a series of evolutionary steps in computing (Foster, Kesselman and Tuecke 2001), with each providing a major advancement, allowing users to solve more complex problems and gain otherwise unattainable results. This evolution started with the single user model, to Massively Parallel Processors (MPPs), to clusters (Krishnamurthy et al 2001), to distributed systems and finally to Grid computing. Recently, large companies such as SUN, IBM and Amazon have been providing Grid solutions by providing resources and services to third parties.

A key ingredient in whether users can utilise grid resources successfully, is the guarantee that users can control their requests. This includes being able to request specific resources, set resource requirements or Quality of Service (QoS) requirements and obtain guarantees that these requirements are met according to their request throughout the duration within which there's an association between the user and the resources.

The focus of this thesis is threefold. First, to provide a flexible and expandable model that is tailored to allowing the user to specify the types of resources they require and the requirements associated with them at a high level, hiding the complexity of the underlying infrastructure and its heterogeneity. Second, to provide a mechanism that allows the selection of appropriate resources according to up-to-date resource information, and finally, a solution that guarantees both the requirements of the user and the resource provider are met throughout the execution of applications.

1.3. Qualities of Service (QoS), Resource Operations and Motivation

There are many definitions for QoS. In this research, *Quality of Conformance* has been chosen which sees QoS as meeting user requirements and specifications. For example, if a user requires a resource with computational power equal to x and the resource provider offers a resource that delivers the amount of computational power required, *i.e. computational power* $\geq x$, provided that the resource does so throughout the time the resource is dedicated to this user, then the resource can be said to have met the request, or *conforms* to the request.

Moreover, each particular domain within the mainstream environment that the user belongs to, will have its own set of QoS requirements and parameters that apply to those domains applications. However, there is a case for carrying the parameters across domains and establishing an arrangement for ensuring that the QoS carried between domains conform to the same definition.

Resource information accessibility is vital to the success of carrying cross-domain requirement specifications and provides the platform for locating the appropriate resources that meet QoS parameters requirements submitted by the user. Current, up-to-date and accurate information relative to each resource ensures that resources selected are offering the level of QoS that is requested by the user and provides the base for creating a working relationship between user and resource provider.

The explanation above presents a problem which needs to be tackled and that is the problem of providing a description of the QoS requirements that can easily be created and used to compare with the level of QoS that a resource is offering in order to carry out appropriate resource selection. There is a need to specify a specific method of describing QoS that the user can use. The information within these QoS descriptions must be extracted in order to carry out resource operations, including resource selection and allocation.

Therefore, resource operations must rely on a user's requirements and the values they set for the level of QoS each resource must provide. Many factors can play a role within this system, such as resource provider policies, user budgets, and resource quantity and time limitations.

1.3.1. Background

Grid computing research has not produced a comprehensive and flexible approach which supports different types of Grids and applications. Current Grid technology is diverse with an inclination for adopting a service oriented architecture (Papazoglou et al 2008) that supports and provides commercial, business-oriented and mainstream services to different domains. However, most current efforts address specific domains such as bioinformatics (myGrid@EBI 2002) or weather prediction, producing specific solutions tailored for applications within those domains and a solution that cannot be carried across to another domain easily.

Moreover, the diversity or lack of QoS support presents major challenges in making Grids a viable tool for the commercial and business-oriented domains where QoS is essential. Many current Grid projects utilise resources that are offered voluntarily, this model cannot be carried forward and the issue of QoS support must be addressed.

The proposed model is driven by the need to find a solution to the problem of flexibility and QoS support and the thesis presents this model and the motivation behind the work. Essentially, it is proposed in line with the assumption that applications from different domains such as, education, engineering and medicine require QoS guarantees and that the requesting of those guarantees is carried out in an efficient manner at a high-level.

1.4. Research Question and Contributions

The selection of resources in Grids is not a straightforward process. This is due to the dynamic nature of Grids, as well as the complexities arising from the distributed and heterogeneous nature of its resources. However, these complexities become more apparent when requesting resources is associated with a specific set of requirements which these resources must meet. The rising number of applications, their types and the different domains they belong to, has meant that the delivery of a standard set of QoS attributes is one of the complexities that needed to be addressed. Moreover, a new approach to QoS specification was needed to be undertaken. This led to the following research question:

Is there a model for QoS of Grids and Grid behaviour that is flexible, capable of carrying out resource operations and is guided by user requirements such that the delivery of QoS to a variety of user types and domains can be guaranteed?

This question can be divided into multiple sub-questions related to implementing any possible model:

At which level must the QoS required be specified within the Grid Architectural Model?

What type of QoS must be supported? How will they be measured?

How can a model that uses this set of QoS be implemented in a way that hides the complexities from the user, while maintaining a successful operational model for resource providers?

How can the issues of local and global resources be addressed? How can the specification of users' privileges to request such resources be addressed?

How will resource discovery and selection occur?

How can reallocation and rescheduling be supported at the same level? And on what basis are these operations triggered?

How can this model be efficiently implemented? What are the components and architecture required?

1.5. Contributions

This thesis presents:

- A comprehensive review of literature on Grid computing, QoS and related projects and models.
- A high-level QoS approach to resource operations driven by a requirements description originating from the user and utilising multiple types of resource information, static and dynamic.
- A dynamic method for calculating specific QoS parameters using up-to-date information, hence increasing the accuracy of resource information, leading to a more accurate resource selection process.
- A multi-tier flexible user model which defines the types of users, their privileges and responsibilities.
- A new method for requesting QoS through specialised interfaces, templates and tier related restrictions.
- A QoS model defining the requirements and communication processes for successful QoS support.
- A resource selection and ranking model for matchmaking resources with the users' description of requirements.
- A method, which employs reallocation, for guaranteeing the level of QoS through the run of an application according to the requirements submitted by the user.
- A novel local approach to searching for resources while maintaining the capability for searching for global resources, using up-to-date resource repositories to hold information on resources that are current. Including, an improved scheduling and reallocation method that employs both the resource ranking capabilities of the model and those of the resource repositories.

1.6. Methodology

The methodology is comprised within the following stages:

- Literature Review:

  Gathering the information that is related to the field of research has helped in formulating the research question as well as providing a clear picture of current research in the field, and the current developments that directly relate to the research undertaken and introduced in this thesis.

- Definitions:

  An important part of answering the research question was that of identifying and defining the required building blocks on top of which this research is carried out.

- Model design:

  The model was designed to provide the platform that answers the research questions.

- Model development:

  The novel model was presented, prototyped, improved, technically developed and produced in detail.

- Simulation:

  The model's evaluation has been carried out using simulation. This is achieved through using a simulation toolkit that has been expanded and extended for the purposes of this research. The simulation toolkit was then used to provide the testing and evaluation environment for this model.

- Evaluation:

  The novel model and its components are evaluated using the newly expanded and extended simulation toolkit. This evaluation process includes:

- Testing the functionalities of individual components of the model
- The success of the QoS model and QoS delivery.
- Evaluating whether the model meets its objectives.
- Evaluating the model in terms of functionality and performance.

1.7. Thesis Organisation

Chapter 2: Presents a comprehensive review of literature and background. It presents a background of Grid Computing, its architecture and its objectives. Next, the chapter introduces QoS and resource operations. Moreover, it also introduces related projects, efforts and research.

Chapter 3: Presents the new model, *Business Grid Quality of Service (BGQoS).* High-level concepts of BGQoS and related definitions are included within this chapter. BGQoS and its associated concepts form the main contribution of this thesis.

Chapter 4: Presents the QoS model implemented within BGQoS. An explanation of the QoS model and the methods implemented within it in order to guarantee the delivery of QoS to the GRC are included within this chapter.

Chapter 5: Presents the components of BGQoS. Complete and detailed explanation of BGQoS components, their responsibilities and specific tasks within the model are included within this chapter.

Chapter 6: Presents the operations employed by the components presented in Chapter 5 in order to carry out their responsibilities. This chapter complements Chapter 5, combining the components with their functional approach.

Chapter 7: Presents the simulation environment and its significance in implementing and evaluating BGQoS. An explanation of the toolkit used and its expansion is included within this chapter.

Chapter 8: Presents a comprehensive evaluation of the important operations, components and functionalities of BGQoS.

Chapter 9: Continues the evaluation of BGQoS. A complete experimental environment is introduced and the results are shown and analysed.

Chapter 10: Presents the summary and conclusion of the thesis. Furthermore, future work and trends are included within this chapter.

CHAPTER 2:

BACKGROUND AND

LITERATURE REVIEW

2.1. Introduction

The last decade of the 20th century witnessed a substantial increase in applications requiring high levels of computing power and network bandwidth. The result was the rapid improvement in hardware, software and network infrastructure. However, the continued development of scientific applications and the thirst in different fields including science, engineering and business (Tserpes et al 2007) to solve bigger and more complicated problems, effectively meant that the technological advancement was lagging and that the current generation of computing at that time which consisted of computers, workstations and super-computers were not enough. Moreover, the fact that these problems were computation and data intensive meant that the resources they required were heterogeneous and often could not be provided within the same organisation and were not located in the same geographical location. This chapter introduces Grid computing, its concepts, architecture and operational model. It also specifically explores work related to this thesis.

2.2. Grids

The availability of powerful computers and high speed networks at a reasonably low cost rapidly changed the computing world. It allowed technology to introduce *resources sharing* such as computational power and storage capacity, as wide area distributed computing models, leading to what is currently known as Grid Computing. By using this new distributed computing environment that allows the user to access diverse types of resources that are located in different places, the users were allowed to solve more problems that require resources that were beyond the capabilities of their own sites, locations or organisations. Moreover, these capabilities were able to provide a reliable method for speeding up the process of carrying out applications. These distributed computing systems are called Grids and will hereafter be called Grids in this thesis. Figure 1 is an example of utilising grid resources in order to assist in product research and development.

The main aim of Grid computing is resource sharing and the utilisation of heterogeneous and geographically distributed resources. This approach has come from different scientific and research institutions and organisations who wanted to carry out compute-intensive and data-intensive applications that required a large number of resources while also requiring them to be completed within a realistic time frame within which the results would still be applicable and viable. However, after Grids established themselves within scientific domains and environments and have

allowed some of the biggest experiments in human history to be carried out, such as the Large Hadron Collider (LHC) at CERN (2011), it was inevitable that Grid Computing would evolve to be utilised within other domains.

Figure 1: Johnson & Johnson on using Grids (OGF 2007)

The inspiration to name these distributed computing systems as Grids is derived from Electrical Grids. Electrical Grids pool together the generation capabilities of a large number of geographically distributed electrical generators to provide usable, reliable, cheap, and universal electrical power. In similar fashion, a Grid is designed around the concept of pooling resources that might be geographically sparse and run by different administrations, in order to provide easy, reliable, standardised, specialised, dynamic and pervasive access to high-end computational resources. This concept has been expanded to include data, instrument and human resources as will be explained throughout this chapter (Czajkowski et al 1998, Roy and Sander 2001).

The project of Grids started with the objective of linking super-computers, combining their capabilities and using them as a single unit. That concept grew to provide a platform from which many applications could benefit, including engineering, physics, data exploration, high throughput computing and service oriented computing. The internet boom, which saw the internet grow at a very high speed alongside the web, has produced interest in exploiting the Web as an infrastructure for running distributed and parallel applications, effectively creating a Grid computing platform (Foster et al 1999, Jeffery 2007).

According to CERN (2011) Grid computing can be defined as a service for sharing computer power and data storage capacity over the Internet. Foster (2002) explains

that for a distributed computing environment to be called a Grid, it must meet a three point checklist:

- It must be able to coordinate resources that are not subject to centralised control: A Grid coordinates resources from different control domains. A Grid enables the sharing of a large number of distributed resources that are not in the same geographical location; otherwise we are dealing with a local management system.

- It uses standard, open, general purpose protocols and interfaces: Grids are built from multi-purpose protocols and interfaces that can address multiple issues such as resource discovery and resource access; otherwise we are dealing with an application specific system.

- It delivers non-trivial Quality of Service: A Grid allows its resources to deliver Quality of Service, meeting user's complex demands; otherwise the potential of the system cannot be guaranteed to be greater than its individual components.

For the effective and correct operation of Grids, the provisioning of system support tools, User Interfaces (UIs), programming languages, programming environments, Grid operating systems, storage services, process management services, security infrastructure and management were necessary. However, the main challenge was that of management of resource sharing and the later challenge of resource scheduling.

2.2.1. Grid Computing Objectives

This section explains the objectives of Grids and the grid computing field in general.

2.2.1.1. Resource Sharing

The main aim for the development of Grid Computing was that of resource sharing (Foster et al 1999), and still remains the main objective. Initially, internal projects were carried out within the same institution, company or organisation. If a project was large and required a large number of resources to be completed within a certain amount of time; more resources were allocated to that project. These institutions would use idle resources within their local environments and make use of them for completing tasks. That opened the door to a situation where certain departments with computationally intensive tasks would be allowed access to idle resources in off-peak hours from other departments or local resources that are under-utilised or idle, to

carry out their operations. For example, if an application needed resources that equalled the combined operational power of a complete floor, administrators would allow those applications to *utilise* the unused computational power during periods when these resources are not used by others in order to successfully carry out the required tasks. This has evolved into the current Grids which allow the sharing and selection of resources or a group of heterogeneous resources such as computing and storage resources.

2.2.1.2. Efficient Utilisation of Idle, Unused and Unallocated Resources

In most organisations there are a large number of unutilised resources. According to IBM (2007), computing resources in most organisations are only utilised to their full capacity and potential five percent of the time. This idle status is by no means limited to CPUs, it also applies to other resources in varying percentages. Grids help organisations pooling their resources together; resources such as computational cycles, software, database servers and network bandwidth.

2.2.1.3. Collaboration

The collaboration between different institutions and organisations is very difficult. Each organisation might have a different architecture deployed. In fact, some organisations might have different deployed architectures between different sites within them. In addition, each organisation has its own policies; guidelines and rules set in place governing any collaboration between different organisations. These guidelines provide the boundaries of operation, and must be adhered to for any collaboration to proceed. A Grid is an environment that allows the collaboration between different organisations, service providers and users. It enables heterogeneous, distributed resources to be pooled together and accessed on-demand. This simplifies access to these resources and makes collaboration possible.

2.2.1.4. Large Problems, Tasks and Applications Solutions

Through Grids, multiple resources can be utilised and pooled together to solve very large problems that would not have been possible if it were not for the access to variable and distributed resources the Grid provides. This has allowed many fields such as weather forecasting and meteorology (Ren et al 2006), industry (Taylor, Surridge and Marvin 2009) and bioinformatics (myGrid@EBI 2002, Desprez and Vernois 2005) to process large amounts of data, run large applications and carry

out computationally intensive simulations. This presented another problem that is also addressed by Grids and explained in the next objective, which is the increased storage demand.

2.2.1.5. Storage Solution

Grids allow the storage of data by providing access to storage resources ranging from high capacity disk storage to long term storage resources. Not only does this provide space for storing data that was not previously available, but also provides the user or application requiring the data access to these resources on-demand (Jeffery 2007).

2.2.2. Grid Features

This section presents the typical features of a Grid (Iamnitchi and Foster 2001, Foster 2002).

- Single Login: The provision of a single login that gives the user secure access to Grid resources. Access control mechanisms are used to control and govern user access to Grid resources.

- Resource Management: The provision of resource management, information services, data storage and data transportation. The highly distributed environment proposed for sharing heterogeneous resources via Grids must be able to meet the challenges of resource management and monitor them through its architecture and protocols.

- Heterogeneity: Grid resources are not of a single type, in the same location or under the same administrative domain. The latter is explained in the next feature of this list. However, the heterogeneity of resource types available in Grids is a vital component and feature.

- Multiple administrative domains: Resources in Grids are located in different locations and operate under the umbrellas of different administrative domains, institutes and organisations. Each of these domains has guidelines, policies and protocols that govern the allocation, usage and utilisation of the heterogeneous resources that belong to them. Grids must therefore operate within these constraints.

- Parallel processing capabilities*:* Results are returned more quickly, efficiently and accurately, using the parallel processing capabilities of Grids. Not all applications can

be modified to run on a parallel computing infrastructure, therefore not every application will be able to function on the Grid. This essentially means that the common belief that Grids will be able to pool resources together to carry out any application many times faster, is not accurate. In fact, developing applications to run on the Grid is both a scientific and engineering challenge (Allen et al 2003), and while there currently exist advanced techniques to do so, they have not been optimised and there is still much work to be done. However, when developed, Grid applications would be able to acquire and release resources according to their needs, on-demand. Applications should also interact easily with users, interact with different types of data and interact with other Grid applications. Grid applications would be capable of completing tasks many times faster than when there was no access to distributed resources. Indeed, some applications may even complete tasks that were not even possible before the resource pooling powers of Grids (Allen et al 2003).

- Dynamicity and Scalability: The Grid by definition is a dynamic infrastructure in which resources can fail, leave the Grid or change according to different conditions. Users, service providers and organisations might also join, leave or change their relationship with the Grid at any point. This is both a feature and a challenge in Grids, one that will be addressed in detail within this thesis. The dynamic nature of Grids enables resources to join the Grid at anytime, leading to an increase in its size that could be significant and potentially affecting performance and other scalability issues. Therefore, Grids must be scalable to accommodate this change, expansion and fluid resource model.

2.2.3. Grid Architecture

Generally, Grids follow a layered architecture which figure 2 illustrates (Foster 2002). Each layer uses the service or services provided by the levels below them and build upon those services. In addition, each level is made of many components which collaborate and communicate between themselves as well as with the lower levels (Ledlie et al 2003, Amin, Von Laszewski and Mikler 2004, MANET Charter 2011).

Figure 2: Layered Grid Architecture (Ledlie et al 2003)

2.2.3.1. Fabric Layer

The fabric layer contains the resources; both logical resources and physical resources, which Grids facilitate access to. Logical resources include distributed file systems and a computer cluster (Foster and Kesselman 1999), Physical resources include computational resources, data and data storage resources and network resources (Foster and Kesselman 1999). This thesis, as mentioned previously, is mainly concerned with computational and storage resources.

This layer defines the interface to native resources, and implements low-level mechanisms that allow the user to access the resources. Once they are accessed, the resources can be used. These mechanisms include but are not limited to resource state inquiries and resource management that must be defined and implemented specifically for the set of resources it interfaces with locally.

2.2.3.2. Connectivity Layer

The basic communication protocols and the core authentication protocols are defined at this layer. These protocols are required for Grid networking service transactions, and provide the mechanism to identify Grid resources and users. Protocols at this layer are derived from the TCP/IP protocol stack. This includes Internet Control

Message Protocol (ICMP) (Rowstron and Druschel 2005), Transport Control Protocol (TCP) (Traversat, Abdelaziz and Pouyoul 2003), Internet Protocol (IP) (Saxena, Tsudik and Yi 2003) and Domain Name System (DNS) (Rikitake 2005).

2.2.3.3. Resource Layer

The resource layer uses the protocols defined in the connectivity layer to control access, negotiation, and initiation, management, monitoring and accounting for Grid resources.

This layer only controls individual resources, without regard to the global state of the system. The resource layer uses the fabric layer (lower layer) to gain access to local resources and controls them. This is done using the information protocol and management protocol. The information protocol is used for calling the fabric layer functions that access and control local resources. The management protocols are used for negotiation and other management of resources (Foster, Kesselman and Tuecke 2001).

2.2.3.4. Collective Layer

The resource layer is only concerned with individual resources. The global state and atomic actions of the complete set of resources pooled together is the responsibility of the collective layer. The collective layer is not associated with a single resource, but is global in nature and is concerned with communication and interactions between selections of resources. Moreover, it is also responsible for the management of these resources. The collective layer is therefore responsible for the coordination between different Grid resources.

The Collective layer is built on top of the *narrow* layers beneath it, such as the resource and connectivity layer. This means that it can implement many sharing behaviour functions without placing extra requirements on the resources themselves and using a limited number of the protocols from the layers beneath it. Directory, co-allocation, scheduling, brokering, monitoring, diagnostics, data replication, software discovery and partner services are in this layer (Netto and Buyya 2010).

2.2.3.5. Application layer

The application layer is the top layer of the Grid architecture. This layer includes the user's applications and enables the use of Grid resources. Created by application programmers, they call the service and protocols provided by the lower layers.

2.2.4. Grid Evolution

In the literature (Al-Fawair 2009), Grid topology evolution is classified into four distinct stages; clusters, intraGrids, extraGrids and interGrid. These stages are illustrated in figure 3.

Figure 3: Grid Evolution (Al-Fawair 2009)

2.2.4.1. Stage 1: Clusters

The initial stage of the development and the grounds for the evolution of Grids was *the cluster*. Clusters are a collection of pooled resources that were used as a unit to provide more computing power when necessary. Clusters are still the smallest and most restricted types of Grids. Cluster computing is built on individual unit processors and commodity operating systems.

Clusters are used to solve computing problems that were proposed by members of an organisation and were beyond the capabilities of a single computing unit. Clusters were implemented locally using the available resources within a single department,

organisation or group. These resources include personal computers (PCs), storage devices and servers in particular, hence creating a heterogeneous pool of resources used for delivering a service that a single unit could not. However, Clusters, as explained before, are the simplest types of Grids, and while they operate in a heterogeneous resource environment, the resource pool itself can only be accessed locally at a single point, using a single queue (MANET Charter 2011).

2.2.4.2. Stage 2: IntraGrids

IntraGrids are distributed systems of clusters within the same organisation or administrative domain. IntraGrids could span multiple geographical locations within an organisation, but in some cases could be a collection of clusters within the same location that are connected together. This allowed organisations to use the basic concept in which clusters of resources are pooled together and expand it into a larger model within the same environment, providing a larger scale of resource sharing and making them available for authorised users. Reliability, security, control over resource access and authentication were the main reason why the concentration of application developers was on intraGrids.

2.2.4.3. Stage 3: ExtraGrids

ExtraGrids open intraGrids to trusted parties and partners, allowing them to share resources and services between each other. These partners are specific and are usually affiliated with the organisation that shares its IntraGrid. ExtraGrids to Grids are what Wide Area Networks (WANs) are to networks. Unlike IntraGrids, the parties that participate in creating an ExtraGrid have differing policies and do not fall under the same administrative domain. However, the relationship between the collaborating ExtraGrids is usually close and mutual.

ExtraGrid can provide a vessel to offload off-peak traffic to a trusted third party, for commercial applications (Crawford et al 2003). Virtual Private Networks (VPNs) are used to make these resources available. Figure 4, from IBM (2007), illustrates ExtraGrid architecture.

Figure 4: ExtraGrid architecture (IBM 2007)

2.2.4.4. Stage 4: InterGrids

The evolution of Grid computing has led to the current generation of Grids. InterGrids (Dias de Assun, Buyya and Venugopal 2008) are a collection of IntraGrid and ExtraGrids that relate in terms of evolution to that of the networking field; from separated Local Area Networks and Wide Area Networks to the inter-networked mesh that is the Internet as we know it. This step of evolution has been as significant for Grid computing.

InterGrids provide the platform for the development of the next-generation of Grid applications that has started to gather pace recently. This has allowed Grid computing to start being introduced into the mainstream. This thesis concentrates on the flexibility and QoS aspects of this evolution phase and is concerned with providing the required conditions for the success of InterGrids.

2.2.5. Grids Classification

Grids could be classified according to multiple criteria, one of which is a classification that relies on their administration purpose and target users. This classification produces four types, National Grids, Volunteer Grids, Project Grids and Enterprise Grids. Another classification is based on the Grids' functionality and produces Computational Grids and Data Grids. All of these types are explained in detail within this section.

2.2.5.1. National Grids

National Grids utilise high-end computing resources as well as data across a nation to create a national computing architecture which is distributed, reliable and integrated. Access to national Grids is controlled by the governments or governmental institutions responsible for it. National Grids were initially restricted to be used for governmental projects, this however has changed recently. Currently, National Grids are also used by educational institutions, research centres and other public sectors (China Grid 2003, D-Grid 2005).

2.2.5.2. Volunteer Grids

Volunteer Grids are an idea in which internet users are given the choice to volunteer unused personal resources. These resources are pooled and used towards achieving a non-revenue scientific, partner or charity goal. In return every volunteer will have restricted access to the Grid. Examples of these types of Grids include Berkley Open Infrastructure for Network Computing (BOINC) (Anderson 2004) and SLINC (Baldassari, Finkel and Toth 2006)

2.2.5.3. Project Grids

Project Grids may span wide areas, potentially across international domains and different organisations that may be located across multiple geographical areas. These Grids pool resources in order to provide service to different communities to achieve a certain scientific or commercial target. Access to these Grids is governed by a privately chosen administrative authority and is usually limited to the organisations that are members of that Grid (Particle Physics Data Grid 2001, Chien 2003, UK e-Science (Grid) Core Programme 2006).

2.2.5.4. Enterprise Grids

Enterprise Grids use the resources located within a single organisation and combine them to produce a powerful, internally distributed computing model. These Grids are at no cost as they only combine available resources within the same enterprise. Moreover, administration is carried out by network administrators within that organisation. Access to these Grids is limited to selected members of that organisation, usually members involved with large projects of importance to that organisation (Cappello et al 2005, Apple 2011a).

2.2.5.5. Computational Grids

A computational Grid is a collection of computing resources. These computing resources represent computing elements and may belong to different owners in different locations and domains. The computing elements themselves might be heterogeneous. The initial purpose of these types of Grids was to run compute intensive applications, in areas where the applications were very large, such as complex scientific and engineering problems. Moore's law states that the processing power of computers double every 18 months. Combining computing elements can provide the users with possibilities that were not feasible before the Grids (Jacob 2003).

2.2.5.6. Data Grids

Data Grids are designed for the storage and replication of data across multiple sites allowing access to this data in an on-demand and efficient manner (Jacob 2003). To illustrate this, the field of medical imaging (Erberich al 2007) is used.

Medical images are substantial in size and considering the number of images taken every day, the issue of data storage needs to be addressed. On top of image sizes, patient information and other related information must be stored with that image. (IBM 2007 and Frost and Sullivan 2007).

The migration from analogue to digital imaging technology has been going on for some time and new imaging technologies such as Magnetic resonance imaging (MRI) and X-ray computed tomography (CT) are used globally. For example the number of CTs taken has grown from 7 million in 2004 to almost 80 million in 2008 (IBM 2007, Apple 2011a). These technology advancements have provided cost-effective alternatives to open surgical intervention and so have been used more regularly and

on a larger scale. Moreover, in parallel, radiologists have started using digital software systems. This has come at a cost, the amount of storage needed to store Imaging data has been growing and is now posing a challenge, not only are there more tests undertaken, those tests have much more data attached to them and that amount of data is growing (IBM 2007, Apple 2011a).

Another challenge is the amount of fixed content which is retained for a long period of time, regularly referenced and does not change. This has increased from 308,000 terabytes in 2003 to 1,250,000 terabytes in 2007 taking up massive amounts of storage in a single geographical location due to the currently used "siloed" architecture (IBM 2007). The volume of data produced by major institutions doubles every six months and there are now around 150 petabytes of medical image related data produced each year (Frost and Sullivan 2007).

Data Grids are responsible for storing the data and providing access to this data to authorized users. Along with the distributed database systems, which can be heterogeneous, they provide the infrastructure that is capable of data storage, data discovery, data handling, data publication and data manipulation.

2.3. Grids in Europe

Over the last decade or so there have been major Grid efforts in America and in Europe as well as elsewhere. The American effort in general defines the Grid as a meta-computing infrastructure, while the European school has concentrated on data. It is worth mentioning that this is not a restrictive statement and there have been both data centred Grid projects in America such as the DataGrid (Chervenak et al 1999) and there have been computationally centered Grid projects in Europe such as EUROGRID (2004).

CoreGrid (2008), the European Research Network on Grid Foundations, Software Infrastructures and Applications for large scale distributed Grid and Peer-to-Peer Technologies, is a major European initiative. CoreGrid has aided in highlighting European research achievements internationally, both in scientific and academic domains. Ultimately, the main aim of CoreGrid was to deliver:

"A fully distributed, dynamically reconfigurable, scalable and autonomous infrastructure to provide location independent, pervasive, reliable, secure and efficient access to a coordinated set of services encapsulating and virtualizing resources (computing power, storage, instruments, data, etc.) in order to generate knowledge" (CoreGrid 2008)

Twenty nine full partners, nineteen countries -eighteen of which are European- have been involved in the project and in achieving its objectives. Six research areas have been targeted, these areas are:

- Knowledge & Data Management
- Programming Model
- Architectural Issues: Scalability, Dependability, Adaptability
- Grid Information, Resource and Workflow Monitoring Services
- Resource Management and Scheduling
- Grid Systems, Tools and Environments

With the LHC (2011) running at the CERN (2011) and producing massive amount of data, Grid computing has been used to store, distribute and analyze 15 Petabytes of data every year, according to Worldwide LHC Computing Grid (2011). A UK project that is associated with the LHC is the UK Grid of Particle Physics (GridPP) (2008). The initial phase of the project lasted for three years between 2001 and 2004 and created a Grid test bed in the UK which was linked to other test beds around the world. This test bed was used to collect real data from different experiments around the world, run by different institutions. The collected data was analysed and used to create, develop and enhance Grid tools and techniques. As of 2008 the third phase of the project was initiated and is expected to last through 2011. The UK GridPP is no longer a test bed and is currently a fully functional Grid infrastructure (Britton et al 2004). It is collaboration between nineteen UK universities, the Science and Technology Facilities Council (STFC 2011), the EU and CERN. GridPP is associated with the following activities:

- Analysing data generated from the LHC's Worldwide LCG project. The LCG (2011) is the worldwide computing grid related to the LHC.
- Sharing experience and expertise with other projects and initiatives in the UK and Europe.
- Sharing experience and expertise with industry in terms of Grid Development and Deployment.
- Working with the UKs E-Science centres (STFC 2011)

Another project that is related to the LCG is the European DataGrid (The DataGrid Project 2004) which was a European Union funded project that concentrated on providing the infrastructure for carrying out compute-intensive operations and analysis on large-scale databases, across widely distributed computing domains. The main achievements of the project were enhancement of middleware stability, successfully deploying it for use by applications, delivering middleware to the LCG production infrastructure and providing euro-wide connectivity. Moreover, it has been utilised by other projects such as UK science program. This project later carried on becoming the Enabling Grid for E-science (EGEE 2010) project which was built over the EU Research Network GÉANT (2011) providing researchers with access to a production level Grid infrastructure. Since 2010, EGEE has been managed by The European Grid Infrastructure (EGI) (2011).

EGI enables access to different types of resources from around Europe. Established in 2010, the EGI provides an infrastructure that allows world-wide multi-discipline collaboration, integrates distributed resources, provides reliable services for computation, data transfer and storage of large data sets and provides the capability of carrying out data intensive and computeintensive simulations and applications faster and in a reliable fashion on top of Grid resources. The main objectives of the EGI are:

- Ensure the long-term sustainability of the European e-infrastructure.
- Coordinate the integration and interaction between National Grid Infrastructures.
- Operate European Grid infrastructure to provide services to different domains.
- Coordinate development and research to enhance European Grids and Grid projects.
- Provide Global services to compliment European services.

- Link European infrastructure with global infrastructure .
- Collaborate and Cooperate with European industry, users and other domains in order to promote the usage of Grid technology.

Other projects have been designed to specifically meet the demands of a domain, such as MediGrid (2005). MediGrid is part of the German Grid Initiative (D-Grid) (2005). this project has been aimed at highlighting the feasibility of using grid services in medicine and life sciences.

The UK NGS National Grid Services (NGS 2011) project was launched to provide researchers across the UK with the resources they may need to complete the scientific goals of their research. The NGS has focused on reliable, robust and trusted services by deploying a common Grid infrastructure that combines resources (nodes) and services from multiple locations.

BEinGRID (Business Experiments in GRID) (2011), financed by the European commission, aimed at identifying business needs that must be met by Grids. The results of the experiments have been carried forward into IT-Tude (2011) which provides a platform for providing services to business and commercial applications through Grid and Cloud computing.

In general, the successful research carried out within Europe on Grids and the services they could provide has been identified and recognised internationally and has provided a base for carrying out further research and achieving advancement in the field.

2.4. Grids for the mainstream

Grid computing went beyond parallel and distributed computing in providing a new dimension of computing that is capable of the management of a large number of geographically distributed heterogeneous resources belonging to different organisational domains. Exploiting these capabilities is no longer limited to the scientific domain. Recently, following a utility based model (Foster 2002, Andrzejak et al 2008), providing resource *via* Grid for the mainstream (Scale Out Software 2011) has been introduced. The following sections introduce this concept, along with the advantages and challenges related to it.

2.4.1. Utility Grid Computing

The next evolutionary step in computing in general and Grids in particular is for mainstream, business-oriented applications, single on-demand usage and individual users to take advantage of the pool of heterogeneous resources, in the same manner that these resources have been able to provide services to scientific applications for the past two decades. Many models for this have been introduced; the most widely accepted approach is that of utility computing (Ross 2004, Yeo et al 2006, Sun Utility Computing 2007, Yeo et al 2010).

The utility computing model (Thickins 2004, Li, Li and Lu 2005) turns resources into services that the customers can pay for according to their requirements included in contracts between them and the providers of these resources. This model is loosely based on the other utility systems of modern life, such as electricity, gas and others. This model offers benefits to both customers and service providers.

There are a large number and range of resources that Grids make available, including computing resources, storage resources and instruments and these can be logically coupled to provide the customer with a platform that conceals these complexities and presents these resources as a single unit. It is of no surprise that the business sector is interested (Middleton 2009). This model promises computing and storage on-demand, cutting the cost of upgrading on a regular basis, amongst the other advantages introduced in the next section of this chapter.

2.4.2. Advantages of Utility Grid Computing

The advantages of Utility Grid Computing are (Foster 2002, Buyya et al 2009):

- Seamless access to computational resources, such as CPUs, as well as to other types of resources, including storage resources and instrument resources, on demand,
- Improving productivity and reducing processing time, according the requirements of the organisations, customers or individual users,
- Provisioning of extra resources, on demand and when required to solve problems that were not possible before. This also eliminates the cost of application specific upgrades, where resources were acquired for a single application and then remained idle without usage, incurring a significant amount of cost and resource under utilisation,
- Utilising idle resources,

- On demand access to a reliable architecture providing an unlimited number of resources that could be accessed if there are unforeseen circumstances,
- If an investment is made, maximum utilisation is insured,
- Access to resource brokers and scheduling techniques that offer a method for selecting the appropriate resources and allocating them to the customer.

2.4.3. Challenges of Utility Grid Computing

The challenges of Utility Grid Computing are (Foster 2002, Buyya et al 2009)

- Customers must re-design their IT related procedures,
- Resource providers must re-design their IT related procedures,
- New policies must be negotiated between customers and agreed upon between customers and service providers, as this model means that users do not have full control over resources as in previous computing models,
- Service providers must understand the requirements of users, in order to agree on a policy,
- Varying quality of service parameters must be provided by the service provider as the users will require support for multiple quality of service attributes. SLAs (Service Level Agreements) are used as contractual agreements assuring the users of the delivery of quality of service by the service provider,
- Financial aspects must be considered and a delivering service, using those financial aspects, must be supported. This mechanism should also support the penalties and compensations required if there's a breach in an established SLA,
- Dynamic and flexible resource allocation is required especially in the context of mainstream applications where requirements (required service and durations) can change dynamically,
- Non-technical issues such as regional, cultural or geographical location issues must be addressed, in some instances requiring the organisations and service providers to change their operational approach and expectations.

2.4.4. Cloud Computing

Cloud computing as a term is derived from "telecom clouds" (Jeffery and Neidecker-Lutz 2010) which indicates a virtualised infrastructure where the end user has no knowledge of the underlying architecture or technology. Recently, more attention has been paid to Clouds as a computing infrastructure when Amazon (2011a and 2011b)

made resources available for their customers, dynamically and on demand. The success of this effort by Amazon has led other resource providers to name their own infrastructures as 'Clouds', the most recent of which has been Apple's (2011b) iCloud.

However, with each resource provider assigning the term 'Cloud' to their resource infrastructure, it has led to multiple definitions of what a C*loud* is. In general, Clouds can be defined as elastic execution environments that provide services by making resources available to users, both internal and external (Jeffery et al 2010). More specifically, clouds are primarily platforms that allow the execution of services and applications across multiple resources in a virtualised environment providing a specific level of service to the user. Virtualisation provides a layer that shields the user from the underlying infrastructure and is important in Clouds.

2.4.4.1. Types of Clouds

Clouds can be classified according to the functionality they provide. Following is an outline of this classification (Jeffery and Neidecker-Lutz 2010):

Cloud infrastructure as a service (IaaS): *IaaS* provides resources as services to users. More commonly known as Resource Clouds, they provide data and storage solutions, such as Amazon's S3 Cloud (Amazon 2011a), as well as providing access to computing resources, such as Amazon's EC2 Cloud (Amazon 2011b).

Cloud Platform as a Service (PaaS): *PaaS* provides computational resources through a platform on top of which applications can be developed and hosted. An example is Force.com (2011) Cloud.

Cloud Software as a Service (SaaS): *SaaS* provides services and applications using the cloud infrastructure instead of providing the cloud infrastructure itself. An example is Google Docs (Google 2011).

2.4.4.2. Cloud characteristics

Earlier in this section, it has been outlined that there is yet to be a clear and standard definition of what a *Cloud* refers to, however, there are common characteristics that are expected to be found in an infrastructure for it to be a candidate to be called a Cloud. Following is a list of these characteristics:

- Elasticity
- Reliability
- Quality of Service
- Adaptability
- Virtualisation
- Multi-tenancy
- Security
- Data management
- Development tools

2.4.4.3. Grids vs Clouds

There has been a debate on whether Grids and Clouds are different and how they differ (Brock and Goscinski 2010). This has led to a question on whether the terms are interchangeable (Foster 2008). However, with no specific definition for Clouds, comparisons have been difficult and have differed in the research (EGEE 2008, Vaquero 2009).

Concept Development

Grids were preliminarily developed for scientific research and domains, with other main stream domains later recognising the benefits that Grids can provide. On the other hand, Clouds were developed for commercial usage from the start.

Overlaps and Common Issues

Resource Grids provide similar services to those provided by Resource Clouds and therefore, there is an overlap between the two (Foster et al 2008). This overlap includes research and common aims. This allows the common usage of concepts, architectures and technological solutions. Virtualisation of resources, scalability, reliability and interoperability are some of the shared aims between the two technologies. More importantly and relevant to this thesis, is that both technologies should ideally be able to guarantee a specific level of QoS.

Grids provide high performance application execution through providing resource sharing mechanisms, while Clouds provide services on demand by providing access to resources that give the impression of a single resource cluster. Both Grids and Clouds

support multiple types of heterogeneous resources and resource types and both provide virtualisation of resources (Brandic and Dustdar 2011), although the extent of vitualisation differs.

Ownership

From the last section we have highlighted that Clouds are systems that provide resources to consumers of a specific party or service provider such as Amazon, rendering their ownership unilateral. Amazon's Clouds (Amazon 2011), Microsoft (Microsoft 2011), iCloud (Apple 2011b) and Force.com (2011) are all unilaterally owned. On the other hand Grids can be unilaterally or cooperatively owned by definition, where the resources shared and provided to a user are heterogeneous in terms or types, location and ownership.

Usability

Clouds are simpler to use than Grids (Jeffery and Neidecker-Lutz 2010), however, in contrast Grids provide users with more information on task execution and underlying infrastructure, as well as greater control in setting their requirements for resource selection. Clouds provide simpler interfaces that restrict the user to a specific set of operations providing a more usable approach to utilising resources. (Vaquero et al 2009).

Domains

Clouds, in their current format, do not cross administrative domains, in contrast with Grids which do. This also explains the simpler security models that are applied in Clouds, as opposed to Grids (Jha and Merzky and Fox 2009) .

Resource management

Grids were the next evolution step after Clusters, where resource management was centralised. Grids provided a decentralised approach to research management. Clouds have both centralised and decentralised resource management capabilities, as long as these resource belong to the same organisation (Vaquero et al 2009).

Virtualisation

Virtualisation is an important characteristic of Clouds, which hides the complexity of the underlying infrastructure and supports a higher level of interoperability by rendering the infrastructure independent. Moreover, virtualisation also provides the capability of providing services to the user, regardless of the actual location of both the user and the resource. While Grids provide a level of Virtualisation that covers both data and computing resources, Clouds offer a higher level of virtualisation by adding the virtualisation of hardware resources (Vaquero et al 2009).

Summary

Grids can be defined as unilaterally or cooperatively owned systems that allow resource sharing. These resources can be heterogeneous and geographically distributed. The selection of these resources depends on their availability and capability and they have been used mainly for satisfying the demands of highly intensive computational tasks typical in scientific experimentation. Clouds are distributed systems that provide access to unilaterally owned, virtualised resources to consumers based on a utility model, with an emphasis on scalability.

2.5. Resource Brokers and Schedulers

Grid brokers and schedulers are responsible for relieving the user from the burden of allocating their tasks to resources and take on this responsibility. Research on scheduling tasks onto appropriate resources is not new and has been an active research area in many computing environments (Katchabaw, Lutfiyya and Bauer 1998, Bobroff et al 2008). A Grid Resource Broker is an integral part of Grids, and is the glue that holds all the pieces together. Brokers are responsible for allocating the appropriate tasks onto the appropriate resources. This task includes other subtasks, such as: receiving user task requirements; resource discovery; task allocation; task monitoring; and result delivery. Moreover, a broker is also responsible for acquiring information on resources, such as resource architecture, availability and other characteristics that are important in determining which tasks run on which resources (Krauter, Buyya and Maheswaren 2002).

Up to the point of the most recent computing infrastructure before Grids, clusters, the most popular method for scheduling was the use of a central scheduling scheme

within a single administrative domain environment. This means the users, the resources and the scheduler are all located under the same administrative domain and the scheduler is given full controlling power over all resources within that domain. Moreover, computational resources assigned by the central scheduler are in the same location as the data and storage resources that hold the data required by the application. Applications are then carried out. This means that all users must submit their tasks to the central scheduler, which then performs optimisation based on achieving higher system utilisation, higher priority service for users and their satisfaction, as well as other criteria that concentrate solely on enhancing the overall performance of the system.

In this work, the concentration is on meeting the user's QoS requirements when running their application on the multi-organisational, geographically dispersed and diverse resource environment or more simply, a Grid. Moreover, the model presented is flexible and could be implemented for different domains.

In general:

- Resource Discovery: The first stage of scheduling which involves the identification of candidate resources. The general filtering process depends on whether the user has access to these resources and whether they are available. However, this filtering process should also take into account the users' requirements. This is elaborated upon throughout this thesis.

- Resource Selection: Once the resources are filtered, the scheduler selects the resource on which the task will execute. This can be done in multiple ways and according to multiple criteria. This selection process is a significant part of the proposed approach within the proposed model and will be explained in detail throughout this thesis.

- Task Allocation: Tasks are allocated to the selected resources from the previous steps. File staging, monitoring and returning the results happens at this stage of scheduling.

2.5.1. Scheduling with QoS

The potential capabilities of Grid computing have been recently highlighted and introduced into the business-oriented domains (NGG 2010), providing an alternative cheaper method for accessing heterogeneous resources that may be geographically distributed and owned by multiple and independent administrative domains. This

distribution has presented an open research field that is directly relevant to distribution in computing and that is the coordination of resource allocation according to user requirements, under the condition that this allocation process is both fair and is in accordance with the policies of both sides. The assignment of applications and tasks to distributed resources, henceforth referred to as scheduling, is the one of the main problems that need to be tackled within the Grid computing.

The dynamic and complex nature of the Grid computing infrastructure not only complicates the scheduling problem, but also justifies the need for QoS introduction into the scheduling procedure to guarantee the requirements of both sides; the user (individual, application or multiple applications) and the resource provider or service provider (Dong and Akl 2006).

Quality of Service requirements and other service driven attributes must be taken into consideration by the broker. Additional or different QoS might be related to Data resources where data services play an important role in data resource discovery. As well as data size and storage location, other data distinct QoS such as access control, modification control and permissions must be considered.

The Grid Scheduling problem is usually viewed as a two-tier problem. The first tier involves the selection of the appropriate resource from the pool of accessible, distributed Grid resources and the second tier involves the allocation of the tasks to be scheduled to the resources selected in the first tier. All scheduling operations are carried out by the broker.

2.5.2. Definitions

Schedulers, in general, can be two types:

- Centralised: Dong et al (2006) define centralised schedulers as those that are based within a single Grid infrastructure; receiving requests by all users and assigning those requests onto the resources in the Grid, accordingly. A Grid scheduler gets a significant amount of help from information repositories that hold information on all resources available for selection. A centralised Grid broker is responsible for the entire resource scheduling process and retains control over all submitted tasks and the resources connected to it. This approach is effective for small infrastructure connected to resources within the same administrative domain. One of the main concerns with this approach is that it provides a single point of failure (Kertesz et al

2007), where all the operations depend on the successful operation of the central broker. More importantly, it could not support or sustain a dynamic environment where users are allowed to specify and submit specific requirements.

- Local: These schedulers are located locally within and perform local scheduling. These distributed entities create a distributed scheduling architecture that aims to utilise resource around the Grid. This approach eliminates the single point of failure as well as providing a scalable solution that can tailor to different environments.

For clarification; the following are a few definitions of terms that will be used frequently over the next few sections:

- A *task* is a single and the simplest part of an application that is to be scheduled, i.e. the tasks are the building blocks of applications.
- An *application* is a complete set of related tasks that when combined provide the definition of an application.

$$\text{App} = \{\text{task}_1 \cup \text{task}_2 \cup \ldots \cup \text{task}_n\}$$

- A *resource* is the unit that carries out the operations on tasks. Resources can be of many types; however resources in this thesis are limited to two types: computational resources, storage resources and networking resources. The latter is the most problematic resource type and is considered to be a bottleneck. Therefore, it requires a larger amount of resource management.
- A *location* is the collection of resources that are governed by the same organisation and are based at the same location.
- A *virtual organisation* is an entity that controls a collection of resources. Multiple virtual organisations could belong to the same administrative domain and adhere to the same policies.
- *Scheduling* is the operation of receiving the request for resources, discovering the appropriate resources, selecting the required resources and allocating tasks onto these resources.

  Currently most scheduling algorithms and models available are *best effort* (Cao et al 2003, Deelman et al 2004, Ma et al 2011) scheduling mechanisms and are insufficient for the demands of mainstream applications. The lack of dynamic and adaptive QoS support in current grid scheduling is the purpose this research has been undertaken.

2.5.2.1. User Satisfaction Driven Scheduling

User driven scheduling has also been explored within Web services (Sun, He and Leu 2007). Some scheduling algorithms adopt the satisfaction of the user as the criteria for their operations and accordingly try to minimize both performance measures as well as cost for the users and their applications. Currently, most schedulers attempt to reduce the time it takes the application between submission and completion. In fact, this period, the *makespan,* is considered one of the most popular performance metrics in Grid computing.

The *makespan* is defined as the time calculated from when the user submits the first task, until the completion of all tasks and the return of the results to the user. Many algorithms have adopted the definition above as a measurement of the performance of those operations and have introduced scheduling algorithms accordingly (Munir, Li and Shi 2007, Selvarani and Sadhasivam 2010). Reduction in the *makespan* came with a higher cost for the consumer, an issue which was not addressed. The users became more aware of this issue and concentration shifted from shortening the *makespan* only to minimising the cost as well as shortening the *makespan.* This led to the introduction of a more dynamic Grid scheduling approach that tailored to both providing the user with performance as well as maintaining an upper limit to how much they wanted to spend (Kim et al 2007, Dong and akl 2006).

Complex scheduling requirements as well as the adoption of the Grid computing environment for mainstream applications is the reason why the introduction of QoS is vital for the success of these applications on Grids and a new scheduling approach is required.

2.5.2.2. Resource Provider Satisfaction Driven Scheduling

Other types of scheduling algorithms aim at providing the resource providers with the best available utilisation of their resources and maximise their economic profit. The utilisation of resources is the amount of time that the resources are allocated to a specific task and are not idle. Some of the major scheduling solutions, such as Condor-G (Condor® Project 2011) adopt this approach as their scheduling criteria.

2.5.3. Broker Types

There are many types of Grid brokers that can be classified according to their task handling capabilities, components and QoS support. Figure 5 illustrates taxonomy of different types of Grid Brokers:

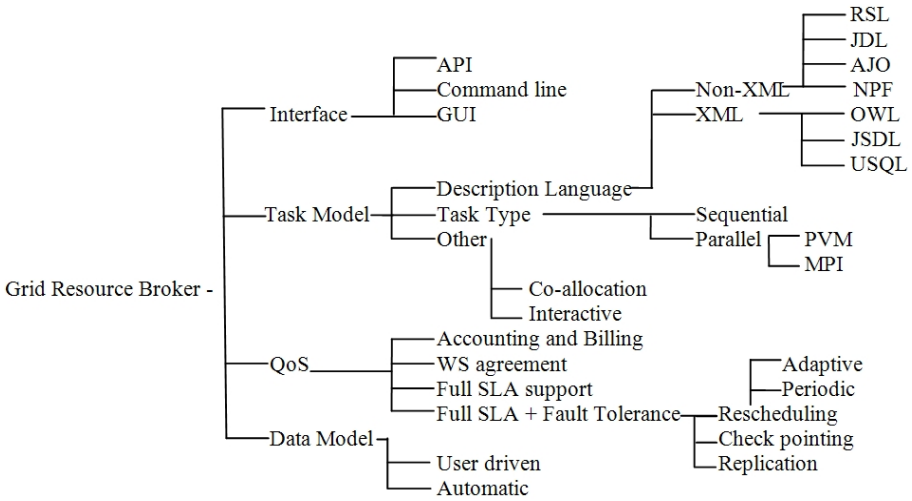


Figure 5: Broker Component Types

2.5.4. Scheduling Models

Resource discovery and scheduling are supported by the scheduling responsibilities of Grid brokers, rendering matchmaking as the main operation for Grid brokers. Figure 6 illustrates the taxonomy for different scheduling models implemented within Grid Brokers:

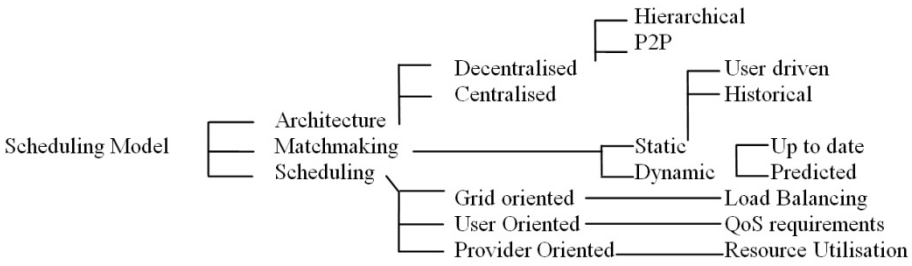


Figure 6: Scheduler Model Types

2.5.5. Multi-Broker Solution

The need for a flexible solution within a Grid requires that this solution be carried across multiple domains and must therefore be capable of dealing with the dynamic and heterogeneous nature of Grids. Currently, available tools and resource management solutions are coupled with specific applications or execution environments, meaning they lack the flexibility sought.

The layered architecture of Grids, introduced earlier in section 2.2.3, means that the flexibility of any solution can target a specific layer for implementation. The first layer, or the resources layer, holds all the software and hardware components. These resources are varied in nature and therefore, no solution is necessary at this level. At the Grid middleware layers there have been some solutions implemented such as UniGrids (2006), however they are restricted to specific operational requirements, in this case between Unicore (2011) and Globus (2011). To tackle this issue, the solution presented in this thesis is implemented at the top levels (the application and collective levels) which provide an entry point to a Grid and its interface with users, as well as taking advantage of middleware services to complement the solution. This high level user driven approach to designing the model has been tailored in order to meet the flexibility requirements while still maintaining a guaranteed level of QoS with a QoS model that can be implemented in different environments.

The solution proposed implements a multi-broker structure which allows different brokers to communicate with each other on behalf of the user once they submit their requirements. The advantage of this implementation is that it allows the expandability of the solution to incorporate different brokers and provides access to different types of resources in different domains to users and applications. In general this allows the brokers to control the operations required, including retrieving user requirements, locating the appropriate resources, submitting tasks and returning results inclusive of related operations such as data movement.

2.5.6. Examples

This section introduces a number of examples in Grid brokering efforts, projects and models:

2.5.6.1. Nimrod/G

Nimrod/G (Buyya 2009), illustrated in figure 7, is a hierarchical system based on computational economy. It is designed for the management and running of parameter studies on computational Grids. Nimrod/G uses 4 adaptive algorithms:

- Cost optimisation
- Time optimisation
- Cost-Time optimisation
- Conservative time optimisation

Figure 7: Nimrod/G (MeSsAGE Lab 2010)

2.5.6.2. Condor-G

Condor-G (Condor® Project 2011) falls in the centralised scheduler category. It is a high throughput centralised scheduling system that allows the user to take advantage of both dedicated and non-dedicated computing resources, which makes Condor G a more complex and different scheduler. The use of non-dedicated computing resources means that a task can be dropped before it is completed in a very heterogeneous environment. This system, developed in the University of Wisconsin provides the following functionalities:

- Task submission to Grid resources
- Submission of task related input/output files and arguments required for task execution
- Retrieving task status
- Cancelling tasks while executing
- Allows users to specify a single location for execution of their tasks
- Reporting back to users, via email feedback on the success or failure of tasks
- Creating a task history log

Condor-G can manage tasks running in distributed locations using a Condor queue and serve as a front end to computational Grids. While Condor allows users to decide at which Grid site to carry out their tasks, if there are many sites to choose from and a decision is not made, Condor-G uses a matchmaking service to decide the Grid site where the tasks are to be carried out.

2.5.6.3. Gridbus Broker

Gridbus (Cloud Computing and Distributed Systems (CLOUDS) Laboratory 2011) is a data Grid broker designed with the main aim of scheduling distributed data Grid applications onto Grid resources. Based on an economical model, it takes into consideration the time and budget constraints of the user when scheduling applications on resources. The Gridbus broker focuses on the scheduling of specific parameter sweep applications with time and cost constraints. This means that the scheduling process is a *greedy* application level process. It has been developed within the Gridbus project, at the University of Melbourne, Australia.

2.5.6.4. Gridway

Gridway (Distributed Systems Architecture Group, Universidad Complutense de Madrid 2010) is a model that performs task submission and execution monitoring. Task execution using this model is a dynamic process that adapts to resource conditions and applications demands for enhanced performance. This is accomplished by providing resource migration capabilities if there is a noticeable performance degradation or resource failure. Figure 8 shows the Gridway architecture.

Figure 8: Gridway Broker (DSAG 2010)

This model has also been used as a method for enabling multi-level hierarchical meta-scheduling structures, where each group of resources is handled as another resource in a recursive manner.

2.5.7. Solution Classification

A comparison between available implemented brokers, with regards to the level they have been implemented at, is detailed in table 1.

Table 1: Comparison between current brokers based on implementation level

Solution	Low-Level	High -Level	
		Middleware	Application
Girdbus	X		
Gridway	X		
GTbroker	X		
JSS	X		
HPC-Europa	X	X	
P-GRADE Portal	X	X	
Koala	X		

2.6. QoS

Users usually would like to specify a set of requirements, guidelines and constraints, collectively referred to as QoS, governing the resource allocation process. In some cases, the user may wish to specify the overall end-to-end QoS, therefore the request for QoS is over all tasks submitted as opposed to the QoS specified for each individual task.

The proposed model employs a resource broker or equivalent entity. Once the broker receives the user's tasks and their requirements, a search is initiated for the resources that are: available; meet the user's requirements; do not exceed the user's constraints, and, are within the permissions of said user. Users that do not set requirements are called *best effort users.* The definition of QoS relates to the agreements between a service provider and their customers which contain a fixed set of well understood QoS requirements requested by the customers.

Quality can be defined from the following three different perspectives and views (Deora et al 2003, Stankiewicz, Cholda and Jajszczyk2011):

- **Quality of functionality:** Quality is considered in terms of the amount of functionality that a service provider offers to the customers. It characterizes the design of an entity and can only be measured by comparing it against others offering similar functionalities.

- **Quality of conformance:** Quality is considered in terms of meeting user requirements and providers meeting their commitments and specification. Quality as conformance, which can be monitored for each service individually, usually requires the users' experience of a service in order to measure the 'promise' against the 'delivery'.

- **Quality of reputation:** Quality is considered in terms of the users' perception of a service in general. This perception is developed gradually over the time of a service provider's existence. Quality as reputation can be regarded as a reference to a service provider's consistency over time in offering both functionality and conformance qualities, and can therefore be measured through the other two types of qualities over time.

2.6.1. QoS in Grid Computing

The heterogeneity of Grid resources, their distributed geographical locations and the different administrative domains they belong to are shared between many users, dynamically and simultaneously. Recently, many business-oriented commercial applications have emerged. These applications can benefit from Grids, especially applications that have high computational and storage needs. The success and failure of these applications to Grids, depends on whether users can be guaranteed that their specific requirements are met or not (Tserpes et al 2007).

This issue becomes more important in the dynamic environment of Grid computing where resources can enter and leave at any time. Resource load and availability vary constantly. The delivery of a guaranteed set of QoS to the user is vital for the success of mainstream applications on Grids.

2.6.2. Related Work in QoS

QoS in networks has been researched extensively, referring to the networks ability to deliver the service asked of it within pre-set guidelines. More specifically, QoS as a concept was a field in which parameters were introduced to measure network operation between two points connected directly together. These parameters include (Katchabaw, Lutfiyya and Bauer 1998) jitter (unwanted variation of one or more characteristics, such as the variation of delay between packets traversing the same route), packet loss (the number of packets that are sent but fail to reach the destination) and throughput (the rate of packets going through the network and reaching their destination successfully). Recently, with the Grid expanding towards different commercial domains (Buccafurri et al 2008, Fölling et al 2010), the QoS of Grids has become an active field of research. The majority of currently existing QoS efforts in Grids concentrate on local optimal QoS scheduling. Although these approaches do take user information and resource information into consideration when allocating resources to different tasks, the concentration is on local resources. Current approaches are not designed to meet Global requirements of QoS scheduling. Our proposed model provides a direct solution to this issue, while maintaining a local approach to scheduling. Golconda and Ozguner compared different QoS based scheduling efforts (2004). Al-Ali et al (2004), propose the Grid QoS Management model (G-QoSM). Their model uses service abstraction in the Open Grid Services Architecture (OGSA) and extends it for QoS properties. G-QoSM reserves quantitative resources, such as CPUs, then allocates and monitors these resources, independently. Another, reservation based approach is presented by Venugopal et al (2008) where they use the alternate offer protocol to make advance reservations. In both these approaches, it is assumed the all resources involved understand the reservation and negotiation protocols.

A quorum based resource allocation and management scheme is introduced by Nam and Youn (2004). Each resource Quorum includes two entities, a middleware entity and a network entity. Both of the entities can satisfy a user's QoS requirements. A heuristic algorithm is proposed by them in order to optimise the performance and cost of every Quorum. Virtual Application Service (VAS) (Keahey and Motawi 2004) is essentially an extended Grid Service with interfaces that deal specifically with the negotiation of SLAs. The main objective of VAS is to ensure that time-sensitive applications are carried out within the time that they are allowed and before a specific deadline, hence the user only needs to provide the time constraint when submitting a request. The system contains application information and application modelling

information that are used to determine the computation resources needed to carry out a task, and reserves them. The General purpose Architecture for Reservation and Allocation (GARA) (Roy et al 2004) is a general purpose architecture proposed. GARA's simple and useful reservation capability has made it popular in the Grid community with its capability to CREATE, MODIFY, BIND and CANCEL reservations. Moreover, it supports flow-specific end-to-end QoS specification and resource monitoring. Curescu and Tehrani (2005) propose an approach where the bandwidth is assigned such that the utility of the whole process, over time is minimised. Ghosh at al (2003) propose QoS optimisation algorithms for allocating resources to tasks in multi-processor environments. Their algorithms pick a QoS reference point, identify the number of replicas required, create the replicas, place the replicas and finally identify the number of processors required in order to maximise overall system performance QoS. Dogan and Ozguner (2004) proposed a solution to allocating individual resources according to multiple QoS requirements (Atakan et al 2006). In their model, the cost of resources is a main factor and is not a constant, but varies during the scheduling process.

A model providing service selection mechanisms based on QoS is presented by Taher, Khatib and Basha (2005). A selection manager is used by Yu and Lin (2005) as a solution for the service selection problem in complex Grid services with multi-QoS requirements. The Selection Manager can be implemented as a combinatorial model or a graph based model. An heuristic is proposed for the combinatorial model based on the algorithms used for solving the multi-option, multi-dimension knapsack problem, also used by Wieczorek et al. (2009), who propose an approach for modelling scheduling problems as an extension to the knapsack problem solution. The graph model, on the other hand, is based on the algorithm proposed as a solution to the multi-constraint optimal path problem (Yu et al 2005). The main objective is maximizing the utility of the system. To achieve this, a utility function is proposed and the algorithm's attempt at maximising this function is intended to increase user satisfaction. Their approach is specifically tailored for the user, without taking the resource provider into consideration. A generalised resource management model is presented by Czajkowski et al (2002) where the Service Negotiation and Acquisition Protocol (SNAP) is used to map resource interactions to platform independent Service Level Agreements (SLAs)

PBS (Altair 2009), LSF (Platform 2009), SUN Grid Engine (Sun Grid Engine 2006) and Condor (Condor® Project 2010) are queuing systems that can be used, efficiently, for delivering a single specific requirement. If all tasks and their requirements are known

in advance, a static approach (Pinedo et al 2005) can create a full schedule for all the tasks at the same time meeting multiple user requirements. Other static approaches include CCS (Hovestadt et al 2003) and GORBA (Suß et al 2005). They both use advance reservations of resources that schedule a sequence of related tasks. However, as they are static approaches, a complete recreation of the schedule is required, if there is a change in resources while executing the sequence of tasks. For example, if a resource fails while execution is in commencement, the whole schedule must be recreated from the beginning. Triana (Taylor 2006, Oinn 2004) , Askalon (Askalon Project 2010), Jopera (Pautasso et al 2004), eXeGrid (Hoheisel 2004) can be used in the development of tools, languages and interfaces used for the composition of workflows, while Pegasus (Deelman et al 2005) and LEAD (Askalon Project 2010) concentrate on supporting the creation of workflows for large scale Grid applications. Taverna (2011) is a system which is concerned with semantic Grid workflows and is implemented as part of the myGrid (2011) project. The aim is to develop sophisticated middleware technology tailored for bioinformatics (myGrid@EBI 2002) in biology and provides fault tolerance solutions and implements a Graphical User Interface (GUI) for the creation and representation of workflows.

In terms of task allocation, in large-scale Distributed Computational Grids, the simple act of submitting a task can be made very complicated by the lack of standards. Some systems, such as the Globus GRAM approach (Czajkowski et al 1998, GRAM 2011), wrap local scheduling submissions but rely heavily on local parameter fields. Ongoing efforts in the Global Grid Forum address the need for common APIs (GGF 2003a). Most often, a user will run Secure Copy Protocol (SCP), File Transfer Protocol (FTP) or a large file transfer protocol such as GridFTP (Allcock et al 2002) to ensure that the data files needed are in place. In a Grid setting, authorization issues, such as having different user names at different sites or storage locations, as well as scalability issues, can complicate this process. Caminero et al (2011) propose a meta-scheduling strategy that takes network characteristics into account. The main objective of their strategy is for it to be scalable and manage QoS in Grid systems. A fuzzy clustering and multi-group classification of QoS for web services approach is implemented by Lin et al (2011) in which a model for marketing and selecting web services based on a multi-group consumer consensus is presented.

The efforts described above have achieved a number of advances in Grid Computing in general, and in resource operations and QoS in particular. Relevant methods have been achieved with success. This work builds on the previous work introduced and incorporates them into it, in order to propose a flexible generic model that can carry

out resource operations for QoS delivery across domains and present an implementation model that could be utilised to accommodate the requirements of these domains.

2.6.3. Projects Related to Market Oriented and Commercial Grid Computing

This section presents a number of examples of efforts, projects and models revolving around commercial Grid environments. These projects reflect the growing expansion trend for Grid Computing into different domains, allowing for many types of applications to be executed. These projects highlight the need for a model that satisfies the general criteria required for resource operations and underline the relevance of this research.

2.6.3.1. GridEcon

GridEcon (2006) is a European funded project exploring the economic challenges in adopting Grid Computing and Cloud Computing. Building a price based model that matches user requests with resources according to the following:

- The quantity of resource units
- The period of time over which the resource is required or available
- The minimum selling price or the maximum buying price
- The expiry date of the request to buy or sell resource

2.6.3.2. SORMA

SORMA (2009) - Self-Organizing ICT resource Management is a funded project that aims to:

- Create a model for realizing self-organizing resource management
- Define an economically sound market structure
- Provide resource users with intelligent tools to access the Open Grid Market
- Provide resource owners with economically sound sustainable and customizable business models

2.6.3.3. FinGrid

FinGrid (2010) is a German research group and consortium of 6 businesses. Its main tasks are to:

- Evaluate the market and compile empirical recommendations and investigate service-oriented Grid cases.
- Come up with different prototypes that are used to demonstrate the feasibility of our concepts in terms of security, accounting, monitoring and pricing.
- Evaluate different types of pricing mechanisms that seem to be applicable for the financial service Grid.
- Propose a solution for how a financial on-demand Grid should utilize both unused resources within a department as well as allow the spontaneous discovery and use of computational resources in other departments or even other organizations.
- Investigate the issues involved for providing support for service level agreements in financial applications.

2.7. Summary

This chapter contained a comprehensive review of literature covering aspects related to this research and this thesis. Grid computing as a concept, its development, architecture, challenges, evolution and projects have all been covered, this was followed by a QoS definition as well as efforts and related work in that area. Finally, examples of current ventures in Grids and Grid computing where the potential user base is broad and covers multiple areas and domains, including those that are business-oriented and commercial, were given. Within this chapter the significance and the relevance of this research relative to current trends in Grid computing has been highlighted. The following chapters of this thesis are dedicated to explaining the environment, approach and methods employed by the novel flexible model proposed in this thesis to accommodate QoS in Grids (BGQoS).

CHAPTER 3: MODEL CONCEPTS AND ENVIRONMENT

3.1. Introduction

This chapter introduces the proposed model BGQoS, its high-level components and its environment. Definitions and the terminology used within this thesis relative to BGQoS are also introduced within this chapter. Moreover, this chapter also introduces the multi-tier user model which has been used within BGQoS. This chapter concludes with an overview and general description of the design and operation of BGQoS, building the foundation for this thesis.

3.2. Commercial and Mainstream Grid Computing

The infrastructure of inter-connected Grids provides the user referred to within this thesis as a *Grid Resource Consumer (GRC)* with the option of acquiring resources that may not be in the same location or administrative domain, as a commodity or utility. More attention has been turned towards providing GRCs with Grid resources, on-demand as utilities, opening the door for a new a paradigm that has picked up pace and has garnered a significant amount of attention within the research community over the past few years, Utility Grid Computing (Ross 2004, Yeo et al 2006).

This new paradigm of Grid Computing has provided a platform for commercial entities to use Grid resources to run their applications, reliably, efficiently and on-demand; however, this has presented a number of challenges. One of the most important challenges is that of guaranteeing the level of QoS promised by resources and another is hiding the complexities of the inter-connected Grid infrastructure from the GRCs. There has been a significant amount of research that has been and is still being conducted on QoS for Grids in scientific communities (Jeffery 2004). However QoS for Grids in scientific communities are rigid and do not provide the versatility required (Middleton et al 2009). It has been noticed that extending these concepts for the commercial, business and personal use of Grids has not been paid sufficient attention (Fölling et al 2010). Moreover, a flexible and general model that is capable of dynamic resource allocation for different types of GRCs within the guidelines and policies of organisations while hiding the complexities is missing. BGQoS attempts to fill that void, by providing a solution for QoS guarantees within this new Grid environment while hiding the complexities that are presented by the underlying infrastructure of Grid computing, as well as being an flexible and expandable model that can provide a solution to multiple domains within the target environment.

3.3. Problem Description

Mainstream, commercial and business-oriented applications would benefit from the access to the heterogeneous pool of resources provided by Grids (BeinGRID 2009). However, this progression within Grid computing coupled with the combination of complex Grid infrastructures, the different locations of resources and the different providers referred to within this thesis as *Grid Resource Providers (GRPs)* has presented a number of challenges that need to be addressed, before a successful integration takes place. Moreover, it is important that both GRCs and GRPs achieve their goals. GRCs would like a specific level of QoS from the resources while attempting to lower the cost of resource acquisition and reduce execution time, while GRPs would like to utilise their resources to their maximum potential while attempting to increase the revenue and impose their specific usage and allocation policies. This may raise a conflict of interest and a mutual understanding between GRCs and GRPs must be reached. This communication process between a resource requester and a resource provider (GRCs and GRPs) is not a simple one and it too raises a number of challenges.

3.3.1. Coordinated Resource Allocation

Current resource allocation and scheduling techniques are diverse, and differ between different domains. Different resource brokers (Krauter, Buyya and Maheswaren 2002) and schedulers are implemented for this purpose and provide an uncoordinated set of resource allocation methods. This diverse approach to scheduling raises the possibility of many problematic scenarios and inefficient ones as well. Moreover, current approaches do not utilise resources to their potential and do not provide the GRC with the service that could be otherwise achieved with agreement. This means that brokers must communicate with the GRC as well as with each other to achieve co-existence and coordination between different Grids. This introduced a new concept called an interGrid where the definition of a single Grid could be expanded to include the collective resources provided by different systems where each could be a single Grid in its own right.

3.3.2. Negotiation

The complex, co-dependent and co-operative relationship between GRCs and GRPs in different fields of computing has been explored, but it has perhaps gained more importance with the introduction of Web Services where the relationship between the two parties had to be redefined. Identification and agreement were required before

the actual service delivery occurs. These agreements are reached via a negotiation process where service consumers request the service they require and the service providers make an offer that the client can accept, turn down or negotiate. Once an agreement has been reached, a contract, in most cases called a Service Level Agreement (SLA) is drafted containing information on the client requesting the service, the service provider and the service being provided. Non-functional requirements, also called QoS, were included in these agreements. This model has been carried forward to Grids and the negotiation process is even more important. The inherent complexity and heterogeneity of Grids makes this a much more difficult challenge. The simplification of an efficient process between GRCs and GRPs is vital and cannot be understated.

3.3.3. Co-allocation of Resources

The co-allocation of resources could be defined in more than one way. It could be defined as the allocation of multiple types of resources belonging to a single provider, to a single consumer, application or organisation. It could also be defined as the allocation of different resources belonging to different owners to a consumer or an organisation. Both of these definitions of co-allocation scenarios could occur in Grids and they must be addressed. The challenge in co-allocation (Li al 2007) of resources not only lies in the complexities related to the different types of resources that are requested and should be allocated, but with the coordination between GRPs to provide the GRC with resource co-allocation capabilities (Netto and Buyya 2010).

3.3.4. Applications

It is assumed that applications for the mainstream to be deployed onto Grids will require the co-allocation of resources described above, in a dynamic and in an on-demand manner. These applications are defined as a multi-requirement, multi-objective sequence of related tasks. These tasks require an execution environment that is created at the top level of the Grid architecture and allows these applications to take advantage of the resources that are provided by Grids. The dynamic nature of application deployment is coupled with the dynamic nature of co-operation between Grids and GRPs.

3.3.5. QoS Guarantees

The QoS requirements of mainstream GRCs are an optional set of requirements that are chosen by authorised users and applications. As a default GRCs will use a *Best Effort* approach (Cao et al 2003, Deelman et al 2004, and Lovas et al 2004) and there have been efforts being made at improving *Best Effort* within Grids (Gallard et al 2008). When they are optional some GRCs may opt for a *Best Effort* approach or may not be authorised to do otherwise. QoS remains an essential, necessary and a vital component of this Grid environment. There are a number of challenges that need to be addressed:

- GRCs must be able to specify their requirements, if they are authorised to do so.
- Appropriate resources must be allocated.
- Applications must be carried out on resources that meet the GRC requirement.
- The GRC must be guaranteed the level of QoS the GRP promise from their resources.
- Appropriate monitoring, feedback, failure detection and reallocation methods must be in place.

3.4. The Model Environment

Large scale Grids are typically composed of a large number of heterogeneous and geographically distributed resources located in dynamic environments under multiple administrative domains and controlled by different organisations and entities. Managing QoS in these environments has become more challenging and relevant because of the recent Grid expansion into business-oriented and consumer-oriented domains (Tserpes et al 2007). Moreover, unlike scientific environments, the GRCs targeted by BGQoS are assumed to be mainstream users and therefore cannot be expected to have knowledge of the required protocols, standards and negotiation processes. This new environment is the focus of BGQoS, moving away from the scientific domains, where Grids, applications and QoS are specific for a certain application and into the more general domain of applications where a specific set of QoS, a scalable and flexible model and standard set of definitions can be used by multiple applications in multiple domains. This emphasises the need for a flexible model, which is one of the main objectives of BGQoS.

Previous work and research has been conducted under the headline of resource discovery where specification languages and algorithms were introduced. Given a resource specification; a resource selection algorithm attempts to find resources using available resource information and the applications attempt to obtain these resources by directly negotiating with resource vendors or managers. These approaches do not provide QoS at a high level, as they only attempt to minimise the makespan or overall completion time of an application (Czajkowski et al 1998, Foster et al 1999, Fahringer et al 2005). The expansion of Grid Computing has introduced new Grid concepts such as Utility Grid Computing (Elmroth et al 2005, Grid Economics and Business Models 2006) where resources can be requested for applications in different domains operating under a utility based model. This expansion means that previous approaches prove inefficient in dealing with resource selection in a more general context and prove inefficient in dealing with preventing and handling failures.

BGQoS aims to provide a flexible QoS-based novel approach with integrated resource discovery, resource selection and resource allocation components. Resource discovery is operated as a GRC request guided search instead of the NP-hard constraint problem used for scientific applications, specifically tailoring resource discovery to high-level QoS descriptions provided by the GRC. The GRC as an entity is responsible for completing the QoS description of the resource requirements phase. Each GRC is placed in our newly proposed hierarchical architecture which can be scaled to a specific domain or organisation and according to their specific policies; each tier of the architecture represents the obligations and authorisation level for every one of the GRCs belonging to it.

BGQoS therefore presents:

- A novel approach for resource selection based on resource discovery *via* the description provided by the GRC for their QoS requirements. The GRC is the focal point which steers the resource discovery phase and is also the main guideline for resource selection. This is done when the resource discovery phase yields a list of resources or resource sets that could potentially provide the GRC with their requirements. However, BGQoS allows the GRC to specify a set of constraints within their description when a request is made. These criteria, along with other domain specific criteria that could be introduced, are used for ranking resources for selection purposes. The highest ranked list is to be chosen, the rest of the potential lists are stored in databases that can be accessed for re-allocation and migration purposes, if necessary.

- A novel approach for reallocation using integrated enhanced stop/start and resource swapping techniques if there is resource failure to contend with and/or there is the situation where the level of specified QoS is not met. This is detected through monitoring the resources and task execution until application completion and session termination.

The resource allocation process within Grid Computing has been generally split into two distinct phases: (1) resource discovery or resource selection; and (2) resource allocation. However, when BGQoS was designed and implemented, a separation between the resource discovery and resource selection operations has been implemented, each defined to be associated with a distinct operational phase. This distinction between the two phases has allowed for the introduction of a more accurate resource operational model. In other words, the approach used within BGQoS decouples the resource specification and discovery process from resource selection, introducing an explicit method for resource selection. This has allowed BGQoS not only to locate the appropriate resources but select the most appropriate in order to execute tasks and applications.

3.4.1. Resource Discovery, Selection and Allocation

The definition of BGQoS entails a duty to identify, select and allocate the appropriate resources to the GRC, by establishing and maintaining communication between them. This involves three steps:

Step 1:

Resource Discovery: Acquiring a list of resources that meet GRC's criteria, these criteria include the types of resources and the level of QoS required. This discovery process uses a database that store resources information and location.

Step 2:

Resource Selection: Once a list of appropriate resources is accumulated, the selection process is initiated. A preliminary filter is initiated followed by the selection of the appropriate resources. BGQoS introduces a novel local selection mechanism that is expandable to a global selection if the local resource cannot meet the requirements of the GRC's application.

Step 3:

Resource Allocation: Once the resources have been selected the tasks are allocated to the selected resources and are executed.

3.5. High-Level Components

The GRCs submit the tasks that are to be carried out using the resources while maintaining a working relationship with the GRPs. The environment and requirements for BGQoS have been introduced earlier, explaining that this environment is tailored to dealing with consumers and applications within a business context, requiring on-demand services from resources using an advertise-select-allocate model that will be introduced within this thesis. Following is a list of these high-level components and their definitions:

3.5.1. GRC

GRCs are the clients that need to run applications using Grid systems and the access to resources they provide, and were referred to as users in our previously published work (Albodour et al 2008, Albodour et al 2010). In terms of the implementation of BGQoS, a GRC should be viewed as a profile which represents a real-life user.

A connected set of tasks form an application. The successful completion of an application is achieved if all the tasks have individually been carried out successfully. The definition of successful task execution is dependent on the application itself and its requirements.

If Tasks $= \{Task_1, \dots, Task_n\}$ make up an application and they are completed successfully, then the application is considered to have been carried out successfully. A GRC states the number of tasks that make up an application prior to submitting the tasks and execution request. This is important for time and cost estimation, introduced later within this thesis.

The necessities of the target environments require that there should be applications of different types, in terms of: Resource Access; i) Local resources only, ii) All resources (local and global) and QoS; i) Guaranteed QoS, ii) Best effort.

A multi-tier GRC architecture has been implemented, with each tier defining the capabilities of each GRC, their access privileges and other information relevant to the

GRCs administrative domain. Two types of GRCs are proposed that fulfil the assumed roles of GRCs within the environment introduced throughout this chapter. Some GRCs will require the maximum allowable QoS parameters that could be specified. Some GRCs require a specific set of QoS. These two GRCs fall under the first type called the *Guaranteed QoS GRCs*. Some GRCs are restricted to best effort options; these GRCs are called *Best Effort (BE) GRCs*. It is worth mentioning that even Guaranteed QoS GRCs can request Best Effort Service.

An objective of BGQoS is to provide *Guaranteed* QoS to authorised GRCs. The guarantee is that their specific requirements are met during the execution of their tasks, and that they are allocated the resources that are capable of doing so. The Guaranteed QoS GRCs have been split into two separate Tiers. The first Tier, which is the top Tier, will have the maximum allowable access to QoS parameters. GRCs belonging to this group will be able to specify a more comprehensive list of the QoS. They are also able to specify cost and time constraints if they choose to do so. The second tier of our GRC model allows users to specify a subset of QoS, predefined by the administrative domain and restricted accordingly. These GRCs will also be able to set constraints for cost and time, however, while the time constraint is still optional the cost constraint is not. This is done to accommodate the multi-privilege reality of any of the domains where mainstream applications apply.

From the discussion above, three layers covering the range from best effort to hard guarantees of QoS are proposed for this thesis. These three layers cover the 3 major types of GRCs and are sufficiently adequate for explaining the operation of BGQoS. However, it is important to underline that since this is a layered architectural model, it is easily expandable to introduce other types of GRC Tiers, according to the specific requirements of every organisation and its structure and policies. This is an important factor within the targeted Grid environment. For every tier, every GRC will be assigned a GRC_{ID} that will be used for identification, authentication and authorisation purposes. GRC tiers within this thesis have been assigned the following naming scheme: Tier A; Tier B; and Tier C. Each of the three tiers of GRCs identifies the privileges of its members, both in terms of resources and the amount of requirements they can set.

Highest Tier (Tier A)

The highest tier of users -users of Tier A- have access to both local and global resources. Global resources can be accessed via a communication process (Bobroff et al 2008) between brokers. This process is explained in chapter 4. Moreover, this tier allows the user to request most specific parameters, larger number of resources and priority reservation.

Middle Tier (Tier B)

Tier B GRCs are able to specify QoS requirements; however, the privileges list of requirements is restricted for users of this tier to a subset of what the top tier Guaranteed QoS GRCs are allowed to specify. The most important differences are however, in that while it is an option for the top tier users to specify cost constraints, it is not optional for this tier of GRC. A GRC at this tier must specify a cost constraint; this is in line with the assumption that in most mainstream application environments, there exists a category of GRCs which has privileges but the cost of those privileges must be administered and limited to predefined budgets.

Lowest Privileges Tier (Tier C)

Tier C is the BE GRC Tier. GRCs at this tier are restricted to best effort allocation (Cao et al 2003, Deelman et al 2004, Lovas et al 2004) of their tasks and applications. However, they must, like users in the second tier, set a cost constraint. This allows their administrative domains to maintain control over cost while still providing the applications with Grid resources on an on-demand basis. No priorities are given to applications submitted by BE GRCs.

All GRCs must first register before using any resources, local or global. BGQoS does this registration locally as it is a distributed model that provides a local scheduling technique to global resources, if required. During the registration period a tier B user and a tier C user must specify their cost constraints, this both simplifies the search for appropriate resources and reduces the overhead incurred by specifying cost constraints, which are mandatory, every time. This constraint can be changed when or if necessary. Once it is, an approval is requested and once approved the new constraint is registered.

Therefore a GRC_b of tier B or a GRC_c of tier C are required to be associated with C_b for time $= t \rightarrow t_i$ and C_c for time $= t \rightarrow t_i$ as cost constraints between time t and t_i .

In order for the registration of any GRC to be accepted, a number of conditions must be met. First, the GRC is checked to belong to the local administrative authority and whether they are of the tier under which they are registering. Second, the cost constraints for the second and third tier GRCs are checked as to whether they are accurate and within the limitations set for each tier. Table 2 shows the operations used within BGQoS in relation to GRCs:

Table 2: GRC Operations

addGRC
Adds a new GRC to the registry.
deleteGRC
Remove a GRC from the registry.
listGRCs
List the GRCs registered within the same organisation and same Tier model.
createTier
Creates a new Tier which GRCs can be assigned to.
deleteTier
Removes a Tier and all its GRCs.
listTiers
Lists the Tier within the same organisation and the same Tier model.
assignTier
Assign Specific Tier with GRC, and map it to the **GRC_{ID}**. Both the Tier and GRC must exist prior to carrying out this operation.
unassignTier
Remove GRC from a specific Tier. Both the Tier and the GRC must exist prior to carrying out this operation and the GRC with a specific **GRC_{ID}** must be assigned to that specific Tier.
modifyTier
Modify Tier related information
searchGRCs
Search for the set of GRCs that belong to a specific Tier. All the GRCs that belong to the Tier specified are returned

3.5.2. GRPs

Each resource is owned by an entity called a GRP. GRPs decide which resources are available to be shared and accessed; at what time, and, for how long. They are also responsible for advertising the characteristics of their resources and registering them to be used. Moreover, they represent the second party in the agreements that are required and created before the GRCs submit their tasks to resources.

3.6. Resources

Resources are defined as a set of software and hardware resources that are controlled by their respective GRPs and belong to a specific administrative domain. Resources can be available for allocation locally or globally. Resources are heterogeneous and geographically distributed, each of which have different functions, characteristics and attributes, delivering differentiated levels of service.

The following section aims at illustrating and defining resources. In the context of this thesis, it is the resources that are relevant to the environment that we study. This is done in order to understand the nature of these resources and the development of BGQoS, resource operations and the matchmaking process. The dynamic and distributed nature of resources and their providers has to be understood in order to facilitate the integration of our methods and components to deliver a functional model. A distinctive approach to identifying resources and describing them is therefore necessary. This section presents resources, how they are described, their properties and the way they affect the resource allocation and matchmaking processes.

3.6.1. Resource Properties

The main properties of Grid resources must be clearly defined in order to provide the relevant information that is used for the correct allocation of tasks.

3.6.1.1. Divisibility

Resources are made of specific units that cannot be split. The processing capacity of a resource, for example, can be split into units with each unit specifically allocated to a GRC. Therefore, these resources must be able to be allocated to a single or multiple

GRCs according to time units. The same can be expanded to bandwidth or storage space.

3.6.1.2. Single or Multiple

A single resource is that which is offered as one atomic unit. The definition of what a *unit* is can be configured to suit each domain, in accordance with the flexibility criteria of our model. For example, a GRP may provide a resource of 10 CPUs as a set of 1 inseparable unit containing 10 CPUs or as a separable unit made of two units the first providing 5 CPUs and the second providing 5 separate CPUs. Multiple units are resources that are offered together, for example, a GRP providing a CPU resource and CPU memory to the same GRC as one offer.

3.6.1.3. Time

Each GRC may have constraints set for when they need the requested resources and the duration of the allocation in accordance with the time they require their tasks to be completed. In addition, a GRP may state in the policies they provide in relation to the offered resource, the time slot for which the resource is available for allocation. The specification of time requirements by both the GRC and GRP facilitate the resource discovery and agreement negotiation processes, as introduced in chapter 6 of this thesis.

3.6.1.4. Cost

The cost of each resource per unit time is used in calculating both the estimated cost for carrying out tasks on application for a specific period of time as well as calculating the required amount the GRC needs to pay at the end of the execution of their tasks.

Each GRC may set a cost constraint, in order to specify the maximum amount they are willing to pay in order to carry out their tasks, factoring in the selection process.

Resource Description

Trading resources based on the GRC requirements for carrying out their applications within mainstream and business-oriented environment is facilitated by providing a viable approach that selects the appropriate resources from the pool of resources that

are offered. This process involves the GRC requesting resources, the GRP advertising them and the broker identifying these resources in order to allocate them to the GRC.

Each resource is associated with a $\text{Resource}_{\text{description}}$:

- Each resource is associated with a $\text{Resource}_{\text{ID}}$ that specifies the resource and where it belongs.
- Each resource has a type that specifies whether the resource is a computational or storage resource.
- Each resource is associated with a set of characteristics; these characteristics specify the capacity, capability and properties of each resource, each measured by a specific unit of measurement.
- Each resource is associated with a set of QoS characteristics which represent the percentage of the full capacity of the resource that the GRP guarantees for a specific time period.
- Each resource is associated with a price per unit of time which is used to identify the cost of using the resource for a specific period of time according the QoS properties of each resource.
- Additional information such as policies and special arrangements and agreements with partners can be included in the $\text{Resource}_{\text{description}}$.
- Each $\text{Resource}_{\text{description}}$identifies whether the unit is a composite resource comprising two or more heterogeneous resources that are offered as one unit, for example CPU and storage space.

Resource Repositories (RR)

Available resources are stored in *Resource Repositories (RR).* Each resource is stored with its description. Some of the characteristics such as *availability* and *reliability* are updated regularly according to dynamically recalculated values based on up-to-date resource information gathered at pre-specified time intervals. The repository is updated dynamically as resources are allocated to different tasks and new up-to-date information becomes available. When resources are registered and stored in the RR, the GRP must decide whether these resources would be available for global GRCs and global allocation.

The collected information is used for creating lists of resource and resource sets that are available at the time of execution to meet the requirements of the GRC and adhere to the constraints set. No list is duplicated. All lists are ranked.

Access to RR

The information on available resources, such as the resource availability, are stored in the RR is vital for the correct selection of resources. The regular and dynamic update of the RR according to current information on resources provides a more reliable decision making platform when selecting resources. Outdated, inaccurate or incomplete information can lead to an inaccurate selection resulting in taking incorrect decisions and failing to meet the requirements of GRCs.

In BGQoS, updating occurs at specific time intervals. The argument is that specifying a time interval to organise updating and prevent "over-dating" reduces the load and overhead, as well as maintaining consistency between the different repositories. These intervals are chosen so that the information is relevant and up-to-date without being too short that they defeat the purpose of reducing the load.

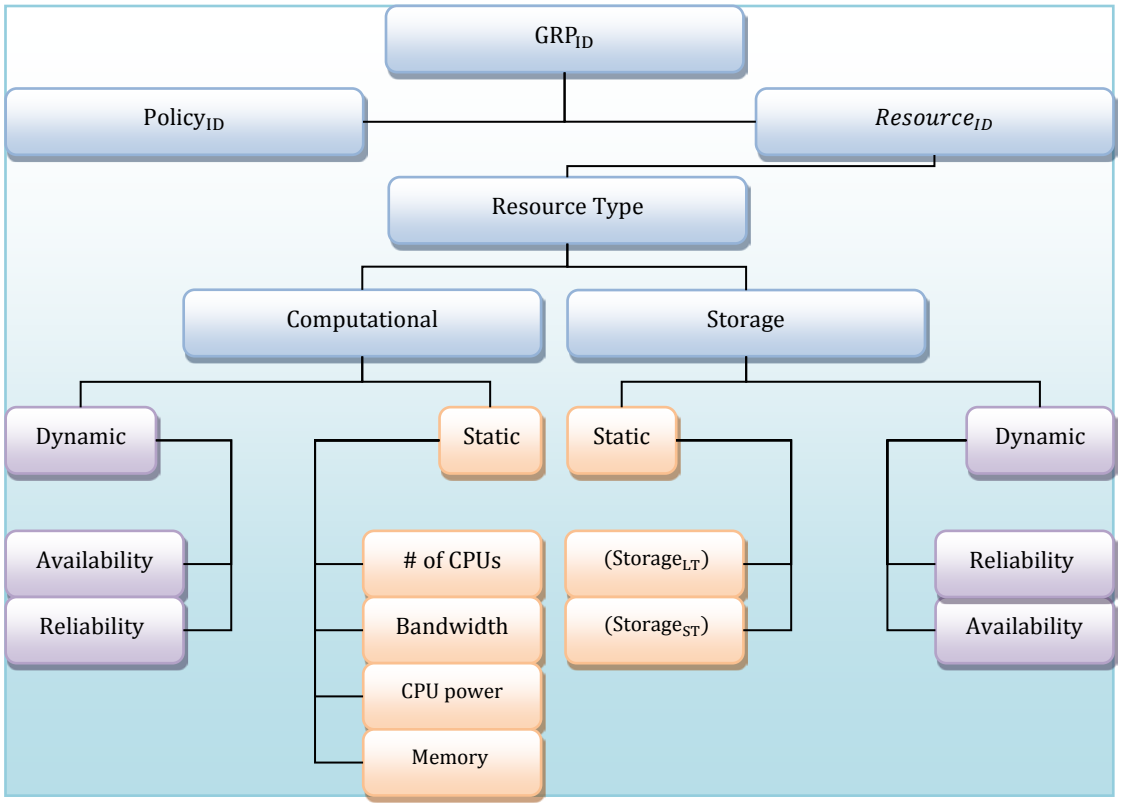


Figure 9: Resource QoS Characteristics.

Resource information provided by the GRP, using the templates introduced within BGQoS, are stored, including information on the resources, their characteristics, their QoS level and the schedule when it is available. Resources might be reserved for a later time but available right now and therefore, the tasks allocated to them must end before the reservation period arrives.

Resource Reservation

Resource reservation is defined under a specific set of parameters that specify the type and period of reservation. Table 3 contains these parameters:

Table 3: Resource Reservation Parameters

Parameter	Description
$\mathbf{R_i^{max}}$	The maximum number of resources required by the tasks that can be reserved.
$\mathbf{t_{start_i}}$	The start time determined by BGQoS
$\mathbf{t_{finish_i}}$	The finish time determined by the BGQoS
$\mathbf{T_i}$	Time Constraint set by the GRC

The flexibility in making these reservations allows BGQoS to utilise the reserved resources more efficiently. For example, if there is a reserved resource with a free interval in between reservations which is sufficient for executing an incoming task, it is considered a candidate resource. The limits for the first three parameters in Table 3 are defined by the GRPs and are controlled according to the policies which apply to each resource set by the GRPs. The fourth parameter in Table 3, i.e. T_i is defined by the GRC according to their own business requirements. BGQoS keeps a high level overview of all tasks and resources.

Multiple Tasks Arriving Simultaneously

A controlled temporal access to the RR is employed to manage multiple requests by multiple tasks arriving at the same time. This ensures that no single resource is allocated to separate tasks at the same time. This guarantees that a single resource is only matched and allocated to a specific task, eliminating the possibility of deadlocks, maintaining integrity and delivering guarantees.

Updating Resource Information

The resource information updating process is carried out automatically. An updating entity is introduced, which is responsible for removing the resource from the list of available resources if it is allocated to a specific task at that time. It is also responsible for updating the dynamic information at the pre-set time intervals. The updating operation implemented within BGQoS replaces similar information about the resources according to a time stamp, with the most recent assumed to be the most accurate. This time interval must be chosen carefully, so as to maintain current information on resources that does not affect the correct resource selection process.

Updating takes place over three steps by the updating entity:

- Collect current information on resources.
- Access the RR and select the resources to be updated, replacing any information that has an older time stamp than the information available from the first step.
- The resource information is updated and the new information is accessible, using a checkpoint and save process.

The drawback to this approach is that it is time consuming, given that the number of resources can be large; however, since this occurs outside the main scheduling process by an independent entity, it is effective.

Resource Related Operations in RRs

The manipulation of RRs is carried out using a set of resource related operations that have been implemented. Table 4 introduces the three operations and an explanations or each.

Table 4: RR Operations

createResource
It allows the creation of a resource, specifying the resource characteristics. It also returns the unique $\boldsymbol{Resource_{ID}}$ assigned to each resource which has been specified before, or created if none was specified. This operation updates the RR and the state of the RR, creating a mapping between the resource IDs and their types and owners.
updateResource
It allows updating the current information on existing resources within the RR. The updating operation can be carried out by both the GRP and BGQoS. The GRP uses this operation if they wish to update resource information for the resources they provide. BGQoS carries out updating operations according to current information retrieved on resources, maintaining up-to-date information in accordance with the objectives our BGQoS. The updating operations are carried out according to the specification introduced earlier within this section and are vital for the correct, accurate and QoS driven resource selection process that has been employed.
deleteResource
This operation allows the removal of existing resources from the RR. This includes, removing the resource information, the mapping information and any other related data connected to the resource. A two phase commit protocol is used in order to guarantee the ACID properties of the database and the transactions carried out on resources and their information; Atomicity (to execute entirely), Consistency (maintain the integrity of the data), isolation (individual transactions run) and durability (the persistence of the result). This means that either the transaction is the “commit” state or will roll back to its original state, and it would not be assigned to a GRC and information would not be updated.

This system of contacting the RRs provides a solution for single points of failure in terms of resource information storage; however it may provide a delay in retrieving information. This is a problem that is to be tackled in future work.

3.7. QoS Definitions

This section introduces the QoS related definitions that are used within this thesis and within BGQoS. These definitions are included in Table 5.

Table 5: QoS Definitions

QoS Parameter
A QoS parameter is defined as a specific GRC requirement, input when submitting an application. BGQoS supports two types of resources, mainly Computing and Storage resources; therefore, most of the parameters that are mentioned hereafter are related to these two types. However, it is important to point out that the model can be expanded easily to accommodate other types of resources as required and relevant to different domains.
QoS Constraints
In BGQoS QoS Constraints have been chosen to have their own heading and are defined as the conditions that need to be met once there are resources that can deliver the level of QoS that is specified by the request for *QoS Parameters.* These constraints, in BGQoS, which for example include *"The latest time that all tasks within the application MUST be completed"*, are delivered from the GRC to the GRP's resources as opposed to QoS Parameters which are delivered *via* the resource description that meets the QoS Parameters of the GRC.
QoS Metrics
QoS metrics are defined as the measurement criteria or units of measurement for *QoS Parameters.*
QoS Characteristics
They can be defined simply as *the QoS parameters* that are provided by resources. Not all resource QoS characteristics are input by the GRP. BGQoS supports, dynamic calculation of specific characteristics, which are updated according to information retrieved dynamically throughout while the resource is available for allocation.
QoS offer
An offer can be described as a response to the input QoS request. These offers are a set of resources that fulfil the requirements input by the GRC. A single offer is part of

the negotiation process.

3.7.1. QoS Resource Management

Resource management according to the QoS requirements of GRCs is the core of BGQoS. This is done through a proposed QoS support model that will be explained in Chapter 4. The components and operations are explained in chapters 5 and 6 respectively.

3.7.2. Application Execution

Once the appropriate resources have been located, BGQoS executes applications accordingly, sending tasks to the resources to which they are assigned. Allocation and execution management components are responsible for the actual task execution that is carried out with the resources that are selected. Chapters 4, 5 and 6 introduce the components and operations required for successful matchmaking. Once the task arrives, the resource allocation components send the task to the appropriate resource, which could be at a different site than where the GRC is located. The information related to a task and required for its execution is downloaded and the tasks are started.

The Task Launcher, which is described in chapter 5, section 5.14, is responsible for starting the tasks on the resources, by creating the appropriate execution environment and retrieving the required files, or in the case where the files are large, pre-scheduling is required in order to make sure the files are available at the right time. BGQoS accomplishes this by first submitting the Task Launcher to the resources instead of the GRCs actual tasks or applications. The Task Launcher's concept was designed for it to run in any environment without modifications or additions necessary for its operation. It is responsible for the input and output files for the application, as well as keeping track of the number of locations where the resources are located, the location of the input files and maintaining a unique $Task_{ID}$ for each task until its completion where the output files are also its responsibility.

The purpose of BGQoS is not only to provide the GRC with the QoS requirements that they request when they submit their mainstream application but to sustain the level of QoS that was promised. The premise that both parties will adhere to what they agree upon is documented in a contract or agreement that is initiated by the GRC, received by the model and offered by the GRP. If there is a violation of the contract, which

might occur for multiple reasons, including resource failure and performance degradation, middleware malfunctions and user errors ("WISDOM" et al 2005, Junqueira et al 2005, Da Costa et al 2007), then the rescheduler in BGQoS is activated and the reallocation process is initiated. However, there are multiple issues to consider before the actual reallocation takes place, these issues are discussed in detail in chapter 6, section 6.9.

In general, an application is within one of the following execution states:

- Tasks pending to be scheduled: At this stage, the tasks are ready to be submitted and their requirements have been identified. The resources are being selected accordingly and the information will be returned.
- Tasks scheduled: Once the appropriate resources have been located, the tasks are scheduled and submitted.
- Tasks queued: This state is not applicable to all applications, and is only applicable if there is a resource where a task is running and the task is scheduled to be allocated to that resource. It is then queued in that resource's queue.
- Tasks running: The resources have been identified, the tasks submitted and are executing.
- Task error: The task does not complete its running phase successfully which could be down to many reasons.
- Task completed: A running task has completed successfully.
- Application complete: All tasks belonging to an application have been completed and therefore the application has been completed successfully. Data placement operations have completed and the execution results are returned to the GRC.

3.7.3. Guaranteed QoS During Execution

In previous work by (Albodour et al 2008), the medical domain has been used to highlight the importance of maintaining a guaranteed level of QoS, and this can be carried across other domains highlighting its importance. More importantly this example demonstrated that giving the GRC the ability to specify requirements and expecting them to be delivered is mandatory if the integration between different domains and the Grid is to be successful. Therefore, BGQoS supports:

- High-level QoS requirements specification.
- QoS metric unification.

- Resource information retrieval and dynamic calculation of relevant characteristics in an accurate and effective manner.
- Negotiation of QoS parameters with GRPs.
- Establishing agreements between GRCs and GRPs.
- Advance reservation capabilities.
- Flexible pricing, based on on-demand agreements that the mainstream GRC can be able to use to acquire the resources they require, when they require them and expect a specific level of QoS.

3.8. Operational Flow within the BGQoS Environment

The operational flow within BGQoS is a combination of two flows, the first from the GRC side and the other is from the GRP side. Figures 10 represent these viewpoints, with a general list of the steps taken by each from the submission of the request by the GRC to the returning of the results of the completed tasks that make up an application. A detailed explanation of each process is introduced in the following chapters of this thesis.

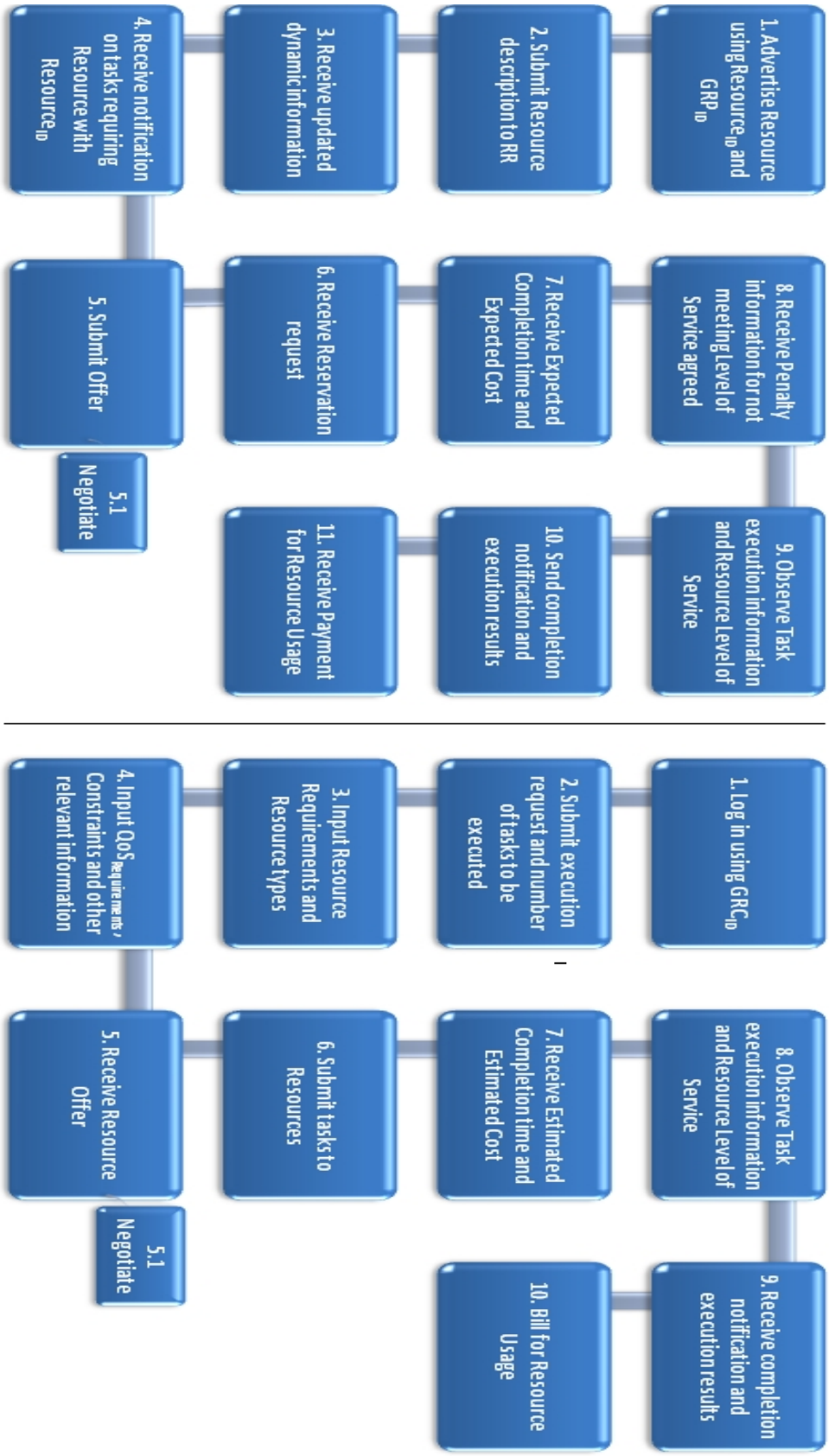


Figure 10: GRC and GRP General Viewpoint

Service Level Agreement (SLA) templates have been introduced in previous research, such as (Schmidt et al 2005 Hasselmeyer et al 2007, Sakellariou et al 2008). BGQoS uses templates in order to simplify the requirements input process and limit it to specific GRCs according to organisational specification. The GRC inputs are converted into an XML-document *via* templates dedicated to every type of GRC relative to their tier. These documents are parsed by the model to retrieve the required information, which includes information on the GRC, their level, the QoS parameters and QoS constraints. In other words, these documents provide the QoS description for the consumer. On the other hand, GRPs provide QoS characteristics of their resources, in addition to the dynamic resource characteristics that are included in a similar XML document as the one related to the GRC. Resource information is stored and can be accessed *via* the Resource Repository (RR). Effectively, these documents represent QoS descriptions for the resources.

BGQoS uses these resource descriptions to locate the appropriate resources to meet the GRC requirements. The GRC and the GRPs exchange those documents, forming the basis of the negotiation process in implemented model. The documents go back and forth between the GRC and the GRP, until an agreement is reached that is acceptable to both parties. This agreement process through BGQoS has many advantages:

- Provides a clear description of the GRCs requirements.
- Provides a clear description of the resource characteristics.
- Simplifies the negotiation process.
- Provides a method for metric unification. The combination of templates and metric unification, reduce the possibility of errors in allocation. In some experimental cases, it eliminated it completely.
- Simplifies the matchmaking process.
- Simplifies agreement establishment.

The agreements reached are the conclusion of the a process initiated by the GRC with providing their requirements, through multiple stages, including resource discovery, resource list generation and resource selection until the negotiation process is completed.

3.9. Summary

This chapter has presented the BGQoS environment, the definitions of related components and the different players within this environment. It has also introduced the concept of a multi-tier GRC model which allows users to be identified with a specific set of characteristics related to a tier, facilitating the inclusion of new GRCs, the expansion of the privileges paradigm and the simplification of GRC related operations. QoS support within BGQoS and the specification and complete description of these agreements, their components and their structure are presented in Chapter 4.

CHAPTER 4: QOS SUPPORT WITHIN BGQOS

4.1. Introduction

The emphasis on QoS within BGQoS is driven by a high-level approach in keeping with the objective of proposing a design for a high-level flexible model that can be carried across multiple domains. Within the environment explained in Chapter 3, there exists a large variety of GRCs, GRPs, and resources. The successful integration between different mainstream domains and Grid Computing is therefore directly related to whether GRCs are capable of requesting specific requirements from the GRPs before utilising their resources. This chapter concentrates on QoS support within BGQoS and explains the methods used to achieve this support. The importance of giving the GRC the ability to state their QoS requirements has been emphasised throughout this thesis. A running example is presented in this chapter to further illustrate the functionality of the QoS model.

4.2. Overall Scenario

BGQoS supports the scenario where a GRC establishes communication with one or more GRPs in order to utilise their resources and the services they provide. This communication process concludes with an agreement that includes definitions and guarantees on the level of QoS, the types of resources and times at which these resources are to be allocated. The high-level design of BGQoS supports the employment of current standards of operation such as SLA specification and provides the basis for QoS establishment between different parties.

4.3. High-Level Abstraction

The operational steps within BGQoS include resource discovery and selection by matching the QoS requirements submitted by the GRC with resource characteristics associated with resources available to the GRC. This matchmaking process may produce a number of resources that can meet these requirements. The QoS related components within BGQoS use $QoS_{parameters}$ as an input and the output is a list of resources that match against the $QoS_{parameters}$.

This provides a high-level abstraction in which the matchmaking process is carried out by BGQoS using a set of $QoS_{Parameters} \rightarrow \{QP_1, \ldots, QP_n\}$ and a set of resources providing the attributes represented as resource characteristics $\rightarrow \{Ch_1, \ldots, Ch_n\}$ meeting the requested parameters as output $\rightarrow \{Set_1, \ldots Set_n\}$.

The simplification of the matchmaking process is an aim of BGQoS, presenting the GRC with the option of attaining the best suited set of resources while hiding the complexities of the infrastructure and differences in definitions in their requirements and the resource characteristics.

If we consider an GRC attempting to carry out a number of tasks by requesting to utilise computational resources, with the following $QoS_{parameters}$: number of CPUs $=$ QP_1, average CPU power $=$ QP_2, and reliability $=$ QP_3, then BGQoS uses the information extracted from the submitted request, which includes the required $QoS_{parameters}$ and maps them to suitable resources according to their characteristics. There are two types of characteristics, static and dynamic. Static characteristics such as number of CPUs are submitted by the GRP within the $Resource_{description}$ in the advertisement phase. These characteristics remain the same while the resource is made available by the GRP and is expected to deliver the QoS specified accordingly.

Dynamic characteristics such as resource reliability are updated at specific time intervals, dynamically and according to current information retrieved from monitoring (Ropars et al 2006) the available resource. The updated information replaces the previous information, while a historical record is kept, providing the model with access to the current state of the resource in relation to specific parameters. The historical record is used for calculating dynamic information such as reliability and availability. BGQoS uses these characteristics to map the requested parameters from the GRC with those provided by different resources available. The computations used to calculate the relative execution time and cost are presented in chapter 6 (6.3) and aim at simplifying the communication process between the GRC and resources by providing the GRC with feedback and information calculated by the model using the information available.

4.4. QoS Offer

At the entry point to BGQoS, the GRC submits an execution request accompanied by a $QoS_{description}$ which includes their requirements, as well as the constraints that complement these requirements. These descriptions are parsed and the $QoS_{parameters}$ requested by the GRC are extracted, these parameters serve as the input to the QoS related components. The resulting output is a list of resources that is

filtered and ranked according to specific criteria. The result is a ranked list of resources that meet the GRC requirements. The top ranked represents the first offer generated to the GRC. This list not only is expected to satisfy the requirements set by the GRC but also satisfies the resource usage policies specified by the GRP.

The ranked list of resources is saved in a database and is referred to if the top ranked list is rejected, i.e. the top ranked list cannot be allocated or does not provide the services required by the GRC throughout the execution of their application. The purpose of the offer is to indicate that the requirements set by the GRC are matched to the characteristics of the resources selected, i.e. a situation of exact match or over qualified, which is explained in more detail in chapter 6, section 6.1. The offer is therefore, dependant on the request submitted by the GRC and is tailored to meet the requirements specified in that request. Continuing with example, if: $QP_1 = 5$ CPUs, $QP_2 = 2.4$ Ghz and Reliability $= 80$ %, and we assume 4 different sets of resources providing the following characteristics:

$Set_1 = \{7, 3.0, 80\}$
$Set_2 = \{3, 2.4, 90\}$
$Set_3 = \{5, 2.4, 80\}$
$Set_4 = \{5, 2.4, 85\}$

Set_2 is eliminated, as it does not provide the required level to meet the parameters submitted by the GRC. The potential list of resources is then$= \{Set_1, Set_3, Set_4\}$. These sets are ranked, with the highest ranked set presented to the GRC as the initial offer.

If we assume that the ranking criteria are solely based on measures to be described by the GRC, the sets are ranked as follows:

$Set_4 \rightarrow Rank = 1$
$Set_3 \rightarrow Rank = 2$
$Set_1 \rightarrow Rank = 3.$

Set_4 will, therefore be presented as a QoS Offer to the GRC.

4.4.1. Offer Generation

The environment for which BGQoS is proposed presents both the GRC and the GRP with an opportunity. The GRCs are provided with the option of utilising resources to carry out tasks that would otherwise be infeasible and the GRPs are provided with an opportunity to conduct business by providing the resources each GRC requires. Realising this infrastructure however, where tasks are computationally intensive and time sensitive, the value of the service by resources may vary and the domain requirements are essential, cannot occur without delivering guaranteed list of QoS.

Within BGQoS, two scenarios may arise, the first is when the communication process is between a single GRC and a single GRP and the second is when the communication is between a single GRC and multiple GRPs. The next few sections of this chapter elaborate on these two scenarios and the QoS support delivered through BGQoS.

The offer generation process provides the GRC with specific resources that meet the requirements and QoS requests and is accomplished by a comparison between the resource characteristics and the QoS parameters and calculating whether there is an intersection between the two sets of attributes. Keeping with the notations, we consider that a GRC specifies a set of n QoS parameters QP = $\{QP_1, \ldots QP_n\}$ that need to be met by the resources that will potentially execute their tasks. Each resource is capable of providing a set of QoS defined by the resource characteristics, where each resource has a set of characteristics Ch $=\{Ch_1, \ldots., Ch_n\}$. Then the intersection between the two provides the offer generated and can be expressed as the following formula:

$$O = QP \cap Ch$$

The formula above simplifies the offer generation process by defining the intersection between the requested QoS parameters by the GRC and the capabilities of resource identified by the resources characteristics, resulting in an offer O.

QP is extracted from the $QoS_{description}$ submitted by the GRC and resource characteristics are retrieved from the Resource Repository (RR), which means that both must be specified in similar formats. The similar format ensures that an

intersection could be achieved between the GRC requests and the resource capabilities.

Every offer is associated with a time limit for which it is considered to be valid. The time validity is the responsibility of the resource broker which examines this parameter and discards offers that have expired.

Let us consider the example again. If 50 tasks are submitted, each requiring a running time of 10 seconds on a CPU at 2.4 GHz and we assume 4 different sets of resources providing the following characteristics and the price for acquiring a resource per unit of time t is represented by P(t) and is measured by price units u.

$\text{Set}_1 = \{7, 3.0, 80\}, \text{P(t)} = 0.8$ u
$\text{Set}_2 = \{3, 2.4, 90\}, \text{P(t)} = 0.5$ u
$\text{Set}_3 = \{5, 2.4, 80\}, \text{P(t)} = 0.8$ u
$\text{Set}_4 = \{5, 2.4, 85\}, \text{P(t)} = 0.5$ u

If we add a Time Constraint T = 120 seconds and a Cost Constraint C = 70 units then the offer generation process can be calculated as:

Step #1:

$\text{Set}_1 \rightarrow$ eT = 80, eC = 64 units
$\text{Set}_2 \rightarrow$ eT = 170, eC = 85 units
$\text{Set}_3 \rightarrow$ eT = 100, eC = 50 units
$\text{Set}_4 \rightarrow$ eT = 100, eC = 50units

Step #2

$\text{Set}_1 \rightarrow$ eT = 80, eC = 64 units
$\text{Set}_3 \rightarrow$ eT = 100, eC = 80 units
$\text{Set}_4 \rightarrow$ eT = 100, eC = 50units

$$O = \{Set_1, Set_4\}$$

Where eT is the estimated time and eC is the estimated cost, both of which are explained in detail in Chapter 6, section 6.3.

4.5. Communication Scenarios

The main concern of the model is to locate the appropriate resources fitting the requirement description submitted by the GRC and assisting the process which concludes with reaching an agreement between a GRC and one or more GRPs for utilising these resources. The establishment of an agreement is necessary before actual task execution using the resources selected. The formation of an agreement between the GRC and GRPs providing them with the resources required is called *negotiation*.

4.5.1. GRC – GRP

The first negotiation scenario represents the situation where a single GRC's requirements request is met by a single GRP, providing the service required. More precisely, a single GRP attempts to deliver the resources with the level of QoS the GRC requests in their description. This process is conducted through matching the resource characteristics associated with the resources with the QoS Parameters submitted by the GRC. Within this scenario, the GRP is expected to deliver an agreed upon QoS in return for a specific price between with a specific time period.

The availability of resources is vital to meet these time requirements and the model uses up-to-date information on the status of the resources and the usage policies attached to resources to determine the availability of these resources when execution is expected to start. A reservation based approach has been traditionally applied to maintain exclusive access to resources in advance. Advance reservation methods are supported within BGQoS.

4.5.2. GRC – GRPs

The second negotiation scenario represents the situation where a single GRC requirements request is met by more than one GRP, hence multiple GRPs must be negotiated with for providing the resources required. More precisely, the model performs the matchmaking operation to select resources using the GRC's QoS description and the resource characteristics. If resources selected belong to different GRPs, it is therefore necessary for more than one negotiation process to take place.

This poses both a problem and opportunity. The problem arises from the management of the negotiation. However, because of the contest between different GRPs in providing their resources, the pricing model changes with GRPs competing against each other. The economic implications, pricing mechanisms and pricing policies are beyond the scope of this thesis and are currently the subject of research. This thesis, instead concentrates on the basis for the negotiation process for QoS.

4.6. QoS Management

The emphasis of QoS Management within BGQoS is on the request made by the GRC for $QoS_{parameters}$ to be used as a standard for locating resources and the guarantee that they are met throughout the execution of the tasks submitted by the GRC.

The GRC request includes a $QoS_{description}$. The $QoS_{description}$ includes the specific QoS parameters and constraints that are required by the GRC, as well as the number of tasks submitted and related information. Each authorised GRC is allowed to specify a set of QoS required parameters, Time Constraint and Cost Constraint.

The Time Constraint, T, identifies the maximum execution time that could be tolerated and is represented by the time period calculated using the start time and the finish time of a specific task. Therefore, the Time Constraint is specified by two parameters:

- t_{start}
- t_{finish}

The Cost Constraint, C, identifies the maximum execution price that could be tolerated and is represented by the unit of currency specified in the template. For example, in our case, the Cost Constraint is specified in GBP (£). Since the pricing mechanism in BGQoS is associated with the period of time that the resources are utilised, the Cost Constraint is defined by a parameter that states the total Cost of running a single task or the complete set of submitted tasks.

- C_{task}
- C_{total}

The QoS parameters within BGQoS are related to the level of the GRC. The layered approach allows each organisation to increase the number of levels or decrease

according to their requirements. Within this thesis a three tier GRC model has been presented, where the top tier or tier A are the most privileged and the bottom tier or tier C are restricted to BE task submission, with only Cost Constraint specification possible and mandatory. These parameters are input through a tier specific interface which mirrors the privileges of each tier upon login.

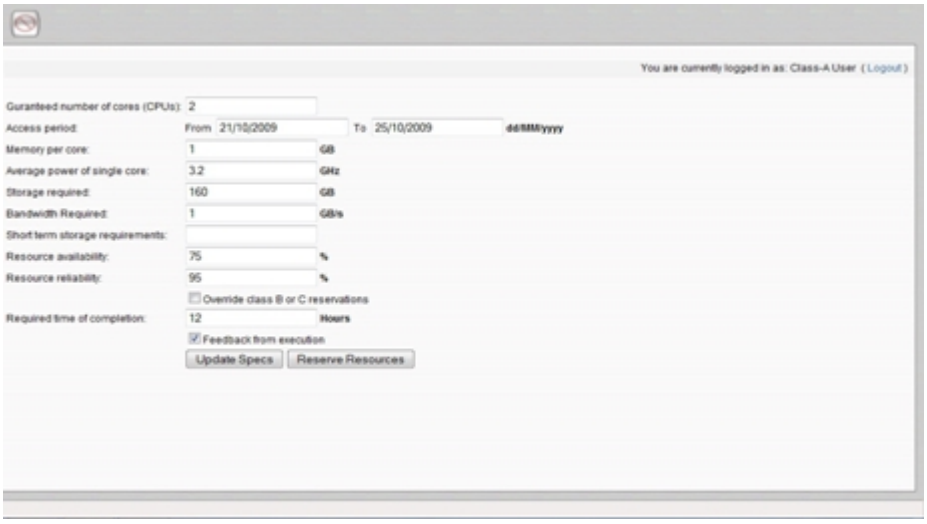


Figure 11: Interface for Tier A GRC

The QoS parameters are converted into an XML based document holding the information that will be used to locate resources, generate offers and establish an agreement. Figure 11 shows the interface for a tier A user and Figure 12 shows an example of a tier A description of QoS requirements:

< GRCs xmlns = "http://my. qos. framework. com/GRCs. xsd" >
< GRC tier = "A" >
< GRCName > classa </GRCName >
< GRC_{ID} > classa </GRC_{ID} >
< FullName > TierA GRC </FullName >
< AccessPeriodFrom ></AccessPeriodFrom >
< AccessPeriodTo ></AccessPeriodTo >
< $QoS_{description}$ >
< QoS unit = "GB" type = "double" name "$Storage_{LT}$" >
< QoS unit = "%" type = "double" name "Reliability" >
< QoS unit = "Kbps" type = "double" name "Bandwidth" >
< QoS unit = "# of CPU units" type = "double" name "individualCPUCount" >
< QoS unit = "GB" type = "double" name "Memory" >
< QoS unit = "GHz" type = "double" name "IndividualCPU" >
< QoS unit = "%" type = "double" name "Availability" >
< QoS unit = "BST" type = "time" name "t_{start} " >
< QoS unit = "BST" type = "time" name "t_{Finish}" >
< QoS unit = "GBP" type = "time" name "C_{total}" >
< OverrideReservations > false </OverrideReservations >
< Feedback > true </Feedback >
</$QoS_{description}$ >
</GRC >

Figure 12: Tier A GRC template

The $QoS_{description}$ definition above contains the following parameters that could be requested by the most privileged GRC, namely Tier A GRCs in BGQoS.

- Long Term Storage ($Storage_{LT}$) → Metric = GB
- Reliability → Metric = Reliability %
- Bandwidth → Metric = Kbps
- Number of CPUs → Metric = #
- CPU power → Metric = GHz
- Memory (RAM) → Metric = MB
- Availability → Metric = Availability %
- Time Constraint → Metric = t = time unit
- Cost Constraint → Metric = c = currency unit

The $QoS_{description}$ can be implemented for multiple tiers of GRCs, according to the requirements and authorisation level associated with each GRC tier. This model can be expanded to accommodate the structure of each domain and organisation.

4.7. Agreement Establishment

Negotiation between the GRC and potential GRPs concludes with the establishment of an agreement that contains information on both parties, the agreed terms and other related information such as the penalties incurred if any violation occurs. Therefore, the negotiation process must be outlined and the protocol explained.

In a distributed heterogeneous environment such as Grids, it is important to specify the QoS requirements dynamically. However, it is unrealistic to expect different GRCs and GRPs to "speak the same language". BGQoS aims to provide a solution that is accessible by different types of GRCs and applications and expandable to different business-oriented or mainstream domains. Therefore, it is important to introduce a negotiation approach that facilitates understanding between different parties involved, be it the GRC or the GRP.

4.7.1. Agreement Basics

The formation of an agreement between the GRC and GRPs providing them with the resources required is called *negotiation*. Current efforts are similar in the negotiation approach. A consumer initiates the negotiation process by submitting their requirements to a Provider. The provider replies with either accepting or rejecting the request.

Work has been done in the area of service negotiation and SLA creation (Hasselmeyer et al 2007, Sakellariou and Yarmolenko 2008). However, there still remains a need for facilitation between different types of SLA templates. In a business-oriented or mainstream environment, GRCs may wish to request specific resources, resource requirements and QoS requirements. Most of the existing work assumes that all parties can understand each other, providing rigid solutions which depend on that assumption.

4.7.2. Agreement Components

The proposed agreement structure can be modified to suit each domain; however, there are common components that represent the spine of any agreement reached between the GRC and the GRPs.

4.7.2.1. Party Description

Each Agreement must contain information on the parties involved. These parties include:

- The GRC: The agreement must contain the relevant information on the GRC requesting the resources, such as the GRC_{ID} and their location.
- The GRP: The agreement must contain the relevant information on the GRP providing the resources such as the GRP_{ID}, their location, the information on the resources provided such as the $resource_{ID}$.

4.7.2.2. Business Relationship

The business relationship portion of the agreement contains information on the application and its execution parameters, such as the penalties and the price.

4.7.2.3. Task Description and Resource Requirements

The types of resources are described within the agreement, including the number of resources of each type. Within BGQoS implementation, the GRC could request computational resources and storage resources. However, this could be expanded in the future to accommodate different types of resources. Task execution requirements are also included within this portion of the agreement, such as the data required for execution, specification of files required and the initiation parameters.

4.7.2.4. QoS Descriptions

The $QoS_{description}$ contains the QoS requirements that have been agreed between the different parties. This description includes the GRC requirements, the expected level of QoS provided and the ratio of GRC acceptance.

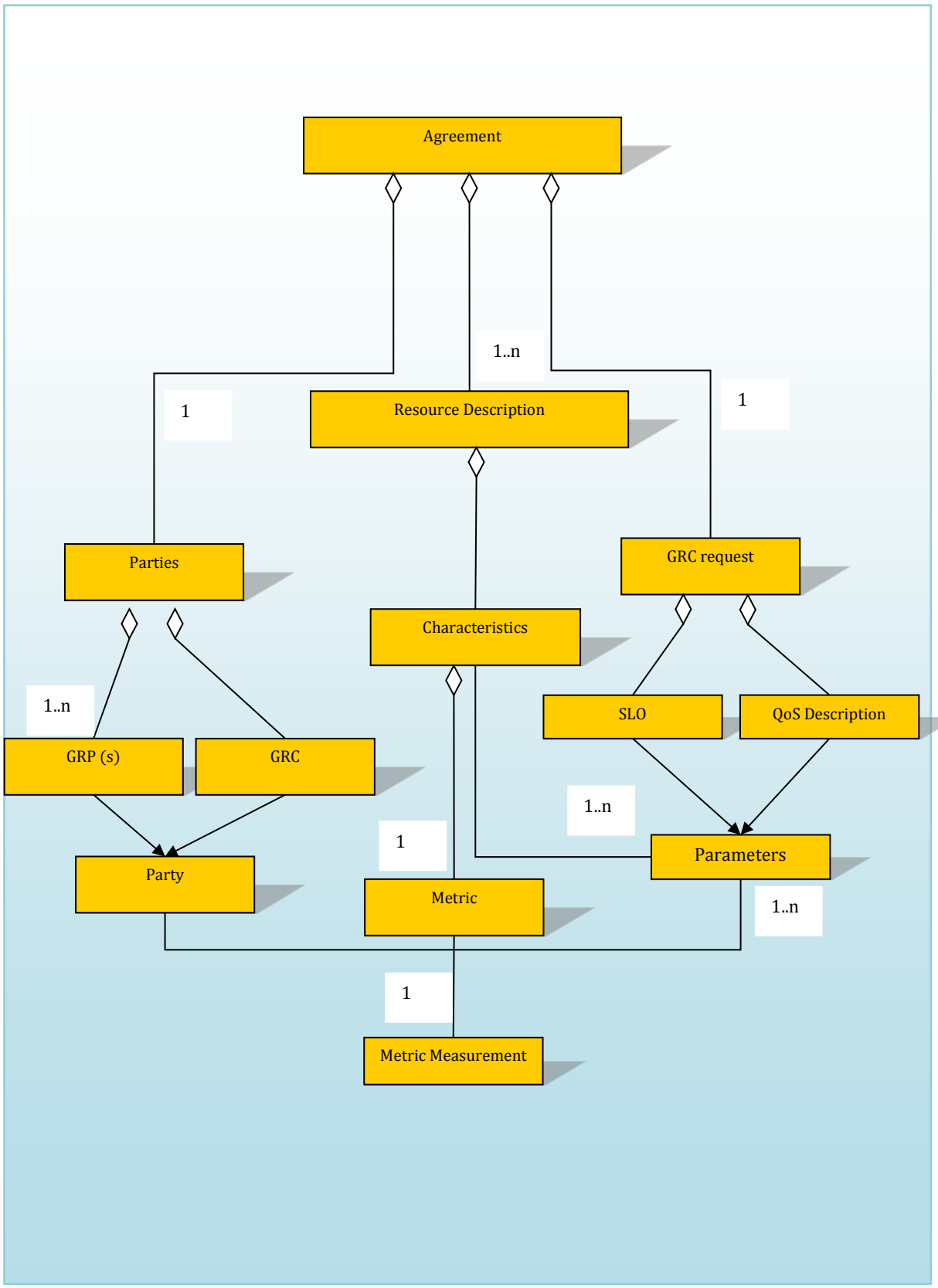


Figure 13: Relationship Diagram (Agreement)

4.7.2.5. Time Constraint and Cost Constraint

The constraints provide the maximum time for completion of the tasks and the budget which must not be exceeded.

4.7.2.6. Service Level Objectives (SLOs)

Agreements are necessary to specify the terms that have resulted from the negotiation. However, within traditional Grid environment assumptions, violations can only occur from the GRP. In reality, the responsibility for an agreement violation can be down to the GRC or the GRP or both. The SLA describes the violation scenario, the responsible party and the actions and the consequences of the violation.

4.8. Agreement Negotiation

The negotiation process is carried out between the different parties involved. Within BGQoS there are three negotiation scenarios that may occur. Following is a description of each scenario:

4.8.1. GRC and Broker

The Resource Broker Component (broker) communicates with the GRC in order to describe the different components of the SLA. When the GRC submits an execution request and the requirements, the parsed information is retrieved by the broker in order to locate the appropriate resources through its various elements introduced in section 5.8.

4.8.2. Broker and GRP

The broker communicates with GRP in order to acquire resources to suit the GRC using the information provided by both parties. In this case, the resource prices are returned to the GRC and if the GRC accepts the cost, a request is made to the GRP for them to accept. If the GRP rejects the offer, the broker communicates with the GRP associated with the second highest ranked list from potential resource sets. The sequence diagram in Figure 14 illustrates the communication scenarios above.

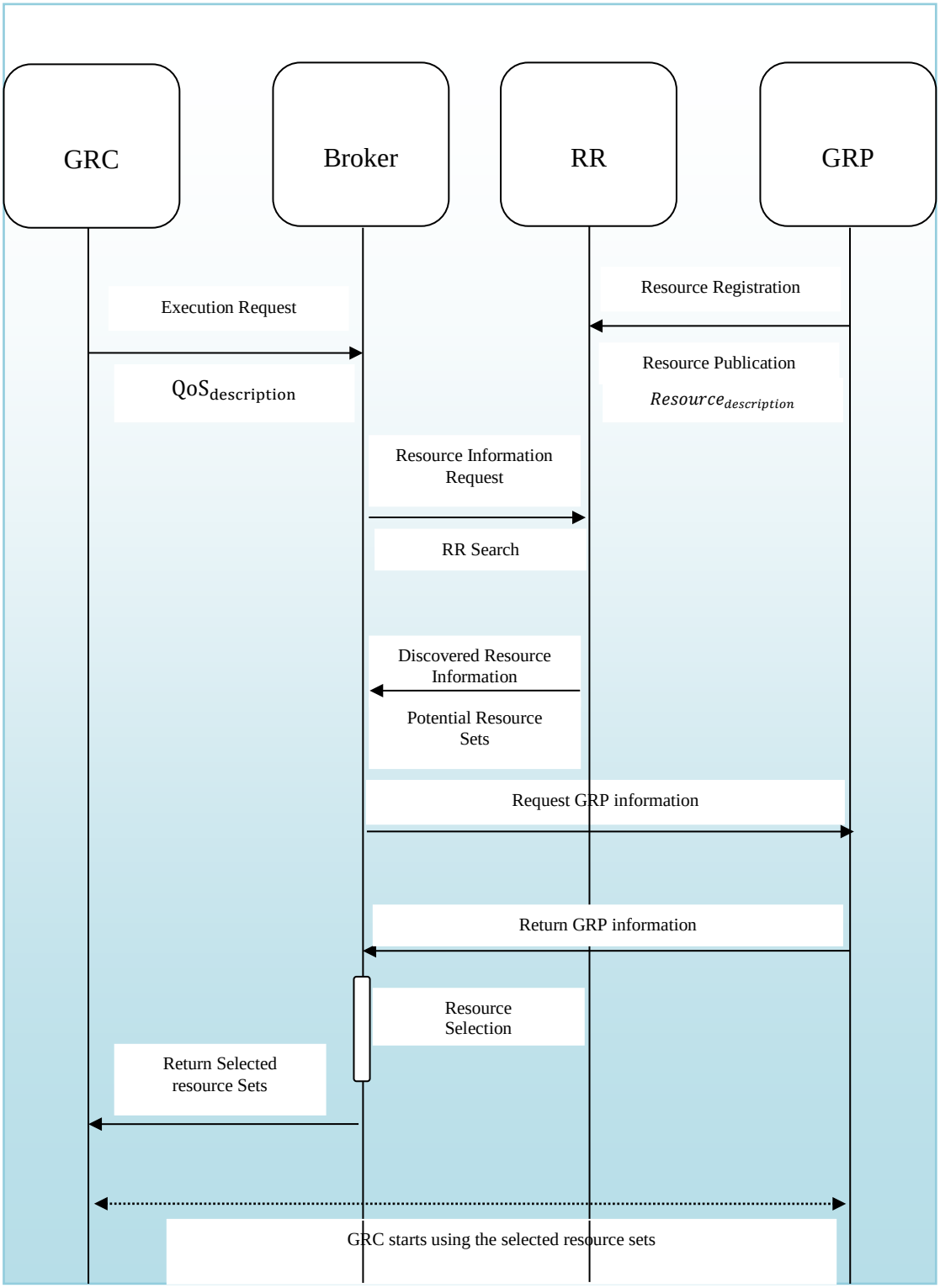


Figure 14: Sequence of Resource Operations between GRC and GRP(s)

4.8.3. Broker and Broker

The broker communicates with other brokers in order to obtain resources that are not located locally. The negotiation occurs at the first broker on behalf of the GRC and the other at the second Broker on behalf of the GRP, allowing them to communicate via the brokers and eliminating any confusion. Moreover, this also allows access to different RRs holding information on the resources available and accessible. This simplifies the negotiation process for global resources, while still maintaining that each GRC's requirement is met and the cost of resource utilisation is returned to the GRP through an agreement set between two different brokers. This is illustrated in Figure 15.

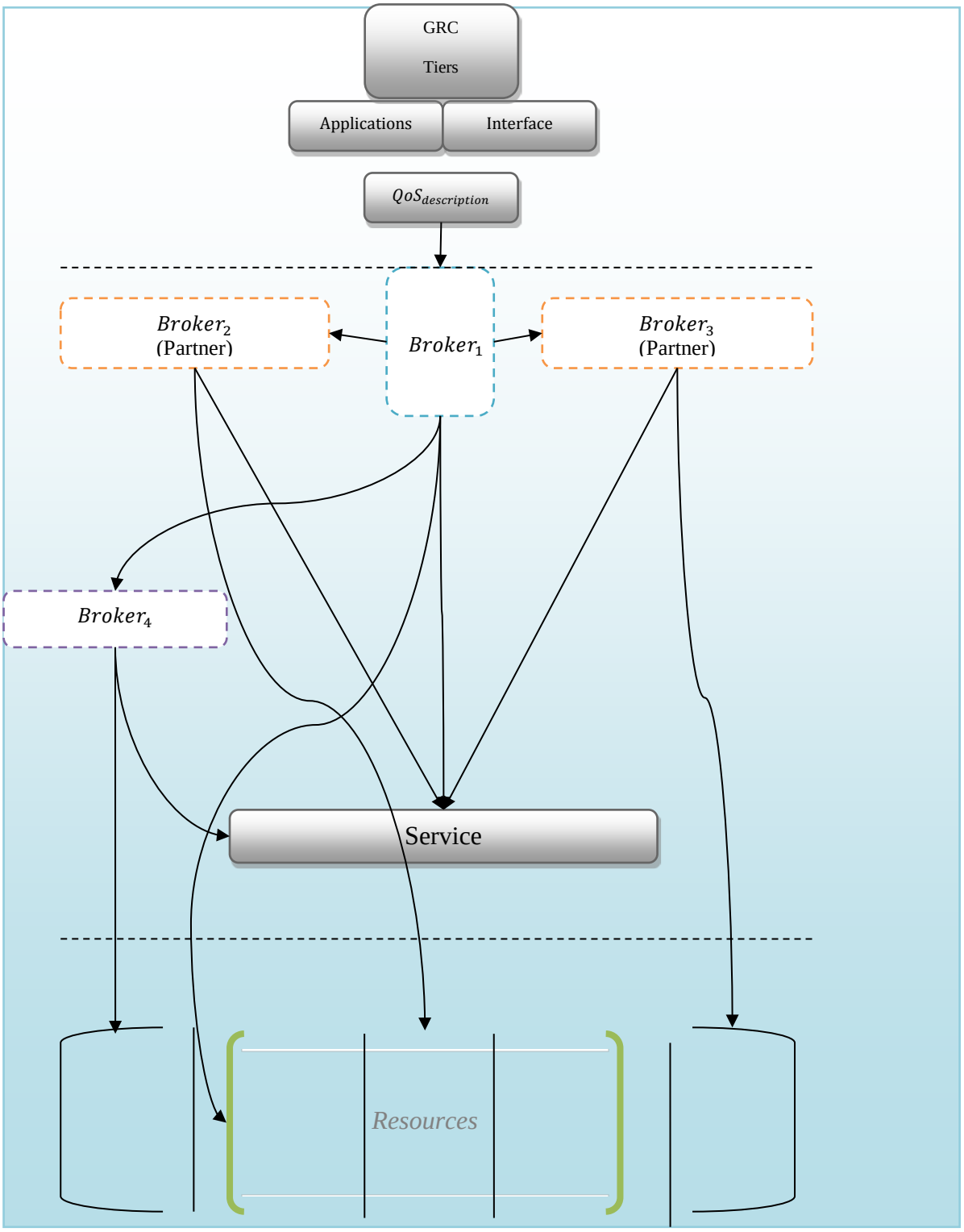


Figure 15: Broker operations and interactions with Partner and Global Brokers.

4.9. QoS Support Methods in BGQoS

4.9.1. BGQoS Flexibility

In the presence of more than one domain and multiple types of resources that may overlap in terms of ownership and description, it is important that the implemented QoS model is capable of distinguishing between different resources, different requests and different types of resources. This is supported and taken into consideration in BGQoS which provides a standard model that recognises these types in which a resource can be assigned a type and associated with a set of specific characteristics and resource information that are identified by a unique ID.

Different types of GRCs are also supported with an expandable multi-tier model in which different assignments can be made according to different domains and different requirements. Moreover, these requests can be designed to reflect each tier, accordingly.

4.9.2. Component Separation

The independence and separation of components from each other and with each associated with a specification enhances the flexibility of BGQoS. In practice this means, that each domain or administrative authority can tailor specific components according to their specification or requirements without affecting any of the other components or the operational functionality of BGQoS

For example, the independence of GRC tier specification and interface design enables the model to accommodate different tier interfaces and the specification of different $QoS_{description}$ generated. This allows the GRC to input their QoS requests using different implementations of interfaces each mapping to the tier that GRC belongs to without altering the operation of functionality of other components. Component separation allows multiple tears to be added with multiple interface designs associated with them, seamlessly and independent of other components within BGQoS.

4.9.3. Symmetric QoS Model

BGQoS employs a symmetric QoS model. Let S be the multi-dimensional space representing the QoS parameters that a set of resources available to the GRC can provide, and let R be the requested $QoS_{parameters}$ by the GRC, which in turn represent a subspace in S. Traditionally, a request (α) has been defined as a subspace in S. An offer (O) is viewed as a point in space S. However, in this symmetric model an offer is considered as a subspace in S just as requests, representing the range of QoS values that a resource is going to supply. In this case O conforms to α if its subspace is within the subspace for α. This interpretation of conformance results in a symmetric model because QoS requests and offers can be specified in the same way allowing for the intersection introduced in the previous section 4.4.

4.9.4. Standardising Request Inputs and Metric Unification

BGQoS is proposed for a multi-domain business context environment, with variable applications and variable GRC populations. It is therefore important that a method that would unify the high-level QoS metrics both for resource requirements and constraints, requested by different applications, is specified.

To illustrate this, we use the constraints as an example. If a specific GRC inputs a requirement that states "all tasks must be completed before 1 PM on Wednesday and execution cost must not exceed 200", it might seem clear enough, however it raises a number of issues. Grid resources by definition are heterogeneous, geographically distributed, belong to different GRPs and adhere to policies within their own administrative domains. Therefore the constraint specification above is not sufficient and must be put in context. 1 PM in which time zone? 200 units of which currency? BGQoS uses templates which resolve this problem. The use of pre defined units at the requirement specification phase reduces the chances of error or confusion.

4.9.5. The Standardisation of the $Resource_{description}$

Resource advertising is an important step in making resources discoverable, and this advertising process includes a description of the resource. A $Resource_{description}$ includes both static and dynamic characteristics that are required in identifying the resource's eligibility for selection in accordance with the input requirements by GRCs. Moreover, these resource descriptions are stored in the RR, and must be ordered according to the specifications of the RR.

Resources could be a part of a candidate set of resources for selection and may have to be compared with other sets of resources through a filtering process, called resource ranking. Resource ranking is a process in which resource sets are compared in order to select the most suited. The success of this process relies on the ability to compare different resources against each other according to their characteristics.

The discussion above highlights the importance of providing a mechanism where resources descriptions could be input, advertised, updated and stored. Moreover, these descriptions should be match-able to GRC requirements and comparable to other resource descriptions by other GRPs.

4.10. Templates

A set of templates have been developed which are responsible for providing the base for descriptions of GRC requirements and resources. These templates can be connected to interfaces that facilitate the input process for descriptions for different types of GRCs and different types of resources, making up an important part of BGQoS. These templates produce XML human readable documents that could be turned into machine readable documents used for resource discovery, selection, allocation and monitoring as well as providing the base for any future agreement between the parties involved. This method not only simplifies the process in which GRCs input their requirements but also makes the matchmaking process faster, more accurate and more efficient within BGQoS. More importantly, it provides the GRC with capability of using high-level definitions to express requirements.

On the other hand, the characteristics of resources are expressed using the same standard approach which allows for precision when matchmaking. The core aim of BGQoS is to deliver a model that can provide a sustained and guaranteed level of QoS delivered by resources to consumers. These descriptions will allow the GRPs to advertise their resources in a manner that is both in line with the requirement specification process of GRCs and is machine readable. These templates are scalable and portable, meaning that they could be expanded for other applications in other domains within the new Grid environment.

4.10.1. Challenges

There are many challenges that have been addressed through the implemented design such as:

- Heterogeneity in resource types and descriptions: GRCs should be able to express their requirements without having the knowledge of how the resources are described by each GRP.
- Heterogeneity in GRP domains: Resource operations should be carried out according to the GRC requirements independently from the domain or GRP types.
- Multi-QoS requests: The GRC should be able to express different requirements as well as the constraints in a single document which can be used in order to perform resource operations.
- Flexibility: The model should be able to accommodate different types of GRCs, requests, domains and advancements in resource technologies.

To address these challenges, the templates introduced represent the canvas from which $QoS_{descriptions}$ are generated supporting a standard method for resource requests, including QoS requirements, constraints and resource types, a flexible set of supported QoS, and an effective method for identifying the metrics under which these QoS have been identified. This also means that it can easily be expanded in order to accommodate different domains. Moreover, the resource discovery and selection process where requirements are matched to characteristics is clarified. In conclusion, a design tailored to accommodate seamless agreement creation using specific terms that accommodate the heterogeneity of the participating parties and environments has been created.

4.10.2. Different Types of Templates

The following types of template have been developed.

- The GRC request
  The specification of GRC requests contains the following:

- GRC information
- GRC tier
- Resource type
- QoS parameters
- Constraints

- Interface related requirements
  Each tier is connected to a specific interface which in turn allows the GRC access to a specific template which they can use to input their request. These interfaces specify the following:

- Search parameters
- Resource requirements
- QoS requirements
- Constraints

- The $\text{Resource}_{\text{description}}$
  Resources and their functionality are defined using templates that include the following information:

- Resource information
- Resource types
- GRP information
- Service association
- Resource capabilities
- Policy information

- The response to the GRC request
  The response to the GRC request includes the following information:

- GRP information
- $\text{Resource}_{\text{description}}$
- QoS parameters
- Estimated time
- Estimated cost

  Figure 16 shows an example of an agreement description, in which the parties are identified and the $\text{QoS}_{\text{description}}$ is included, as well as the QoS requirements for each parameter:

< $SLA\ name$"$example213$" $xmlns$ = "$http://my.qos.framework.com/GRCs.xsd$" >
< $Parties$ >
< $GRC\ name$ = " $Class - A\ GRC$" />
< GRC_{ID} = " Tier − A GRC" />
< GRP_{ID} = " " /
< $GRP\ name$ = "GRP_{name}" />
</$Parties$
< $AccessPeriodFrom$ > 21/10/2009 </$AccessPeriodFrom$ >
< $AccessPeriodTo$ > 25/10/2009 </$AccessPeriodTo$ >< $QoS_{description}$ >
< $QoS\ unit$ = "GB" $type$ = "$double$" $name$ "$Storage_{LT}$" >
< $QoS\ unit$ = "%" $type$ = "$double$" $name$ "$Reliability$" >
< $QoS\ unit$ = "$Kbps$" $type$ = "$double$" $name$ "$Bandwidth$" >
< $QoS\ unit$ = "GB" $type$ = "$double$" $name$ "$Memory$" >
< $QoS\ unit$ = "GHz" $type$ = "$double$" $name$ "$IndividualCPU$" >
< $QoS\ unit$ = "%" $type$ = "$double$" $name$ "$Availability$" >
< $QoS\ unit$ = "BST" $type$ = "$time$" $name$ "t_{start} " >
< $QoS\ unit$ = "BST" $type$ = "$time$" $name$ "t_{Finish} " >
< $QoS\ unit$ = "GBP" $type$ = "$time$" $name$ "C_{total} " >
</$QoS_{description}$ >
< $Agreement$ >
< $SLO\ name$ = "$SLOobjective$"/>
< $Guarantor$ > GRP < $Guarantor$/>
< $Resource_{description}$ >
</$Resource_{description}$ >
< $AcessPeriod$ >
< $AccessPeriodFrom$ ></$AccessPeriodFrom$ >
< $AccessPeriodTo$ ></$AccessPeriodTo$ >
</$AccessPeriod$ >
< QoS >
< $IndividualCPU$ >
< $Value$ > 3.2 </$Value$ >
</$individualCPU$ >
< $Memory$ >
< $Value$ ></$Value$ >
</$Memory$ >
< $Storage_{LT}$ >
< $Value$ ></$Value$ >
</$Storage_{LT}$ >
< $Bandwidth$ >
< $Value$ ></$Value$ >
</$Bandwidth$ >
< $Reliability$ >
< $Value$ ></$Value$ >
</$Reliability$ >
< $Availability$ >
< $Value$ ></$Value$ >
</$Availability$ >
< $time\_start$ >
< $Value$ ></$Value$ >
</$time_{start}$ >
< $time\_finish$ >
< $Value$ ></$Value$ >
</$time_{finish}$ >
< $Cost_{total}$ >
< $Value$ ></$Value$ >
</$Cost_{total}$ >
</QoS >
</$Agreement$ >
</SLA >

Figure 16: SLA Template

The information extracted from these XML documents is used to carry out the resource operations and monitoring operations within BGQoS.

4.11. Summary

This chapter has explained how BGQoS supports QoS. This support is essential and requires that there exists a specific communication process between different entities producing a working relationship and agreement between the different parties envolved. This chapter has explained the different communication partners, their communication process and the communication requirements. The combination of the different operations, definitions, modules and communication presented within this chapter, make up the QoS support within BGQoS. The next chapter explains the different components that carry out the support.

CHAPTER 5: BGQOS SYSTEM COMPONENTS AND DESIGN

5.1. Introduction

BGQoS aims at presenting a solution that is: flexible so that it can be expanded into multiple domains; QoS driven so that it meets the requirements set by different types of GRC, and, complete in that it covers the multiple aspects required.

The entities and components required to implement the model are the focus of this chapter. Initially, the full list of components is introduced which is followed by an explanation of where each fits within the Grid architecture. Then a detailed description of each component is given. In the implementation of BGQoS, advantage has been taken of current solutions and available technology, and new solutions and approaches have been added.

5.2. Model Layers

BGQoS includes capabilities for specifying a selected set of QoS, which can be expanded in future work to include more such as security and provenance. Moreoever, BGQoS provides the GRP with the control over their resources by enabling specification of usage policies and price requirements. Furthermore resource management solutions are included in the scheduling and allocation process. BGQoS is therefore a comprehensive model where the main focus is not the scientific domains, but more in the mainstream. The flexibility and scalability of the model means that as a solution it could be implemented in multiple domains with each domain specifying their own requirements and definitions. In addition to providing a solution to QoS specification and resource matchmaking, the model employs a flexible, expandable and multi-tier GRC architecture that could be tailored to each specific domain or organisation. The simplification of the matchmaking process, the clarification of specifying QoS requirements and the multiple venues in which GRCs can obtain information on their task have been included for a more complete model. Figure 17 illustrates the different layers of BGQoS and the operation contained within each layer.

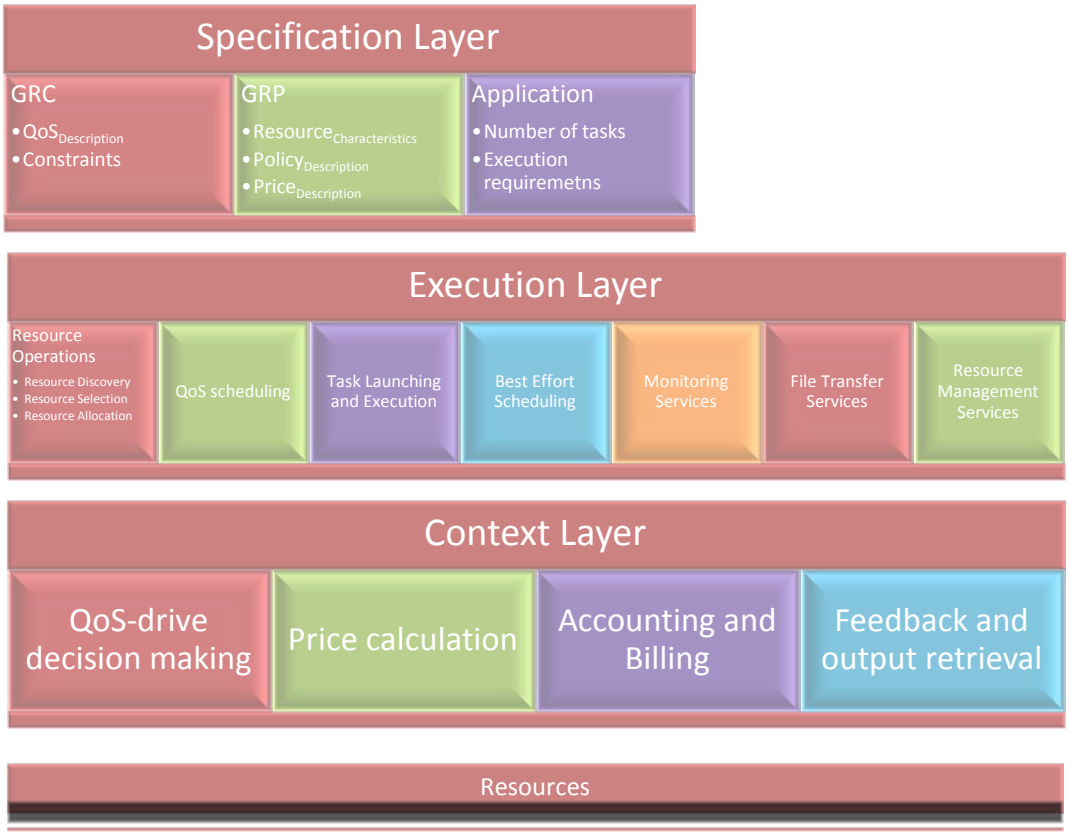


Figure 17: BGQoS Layers

Dynamic calculation of resource parameters and updating a database holding this information provides up-to-date accessible information that can be used for more accurate and more efficient resource discovery and selection processes to take place. This information is updated on a regular basis and is used for guaranteeing that the level of QoS provided by resources matches that which the GRC requires

The fault tolerance and QoS recovery mechanism involved provide a guarantee that the level of QoS over the run of the application and the tasks within this application are not compromised and that the GRC is guaranteed the promised level of QoS. If there occurs a situation where this is not possible, GRCs are informed and the appropriate measures taken in accordance to prior agreements that are in place before the execution commences.

Different components within the model ensure that the tasks are executed on the resources and the results of the successfully completed tasks are returned. Moreover,

the billing components of the model make sure that the rights of all parties involved are maintained throughout the execution phase.

5.3. Implementation Components Overview

The rest of this chapter is dedicated to the detailed explanation of the different components that are implemented. The components that collectively make up BGQoS are introduced in Figure 18.

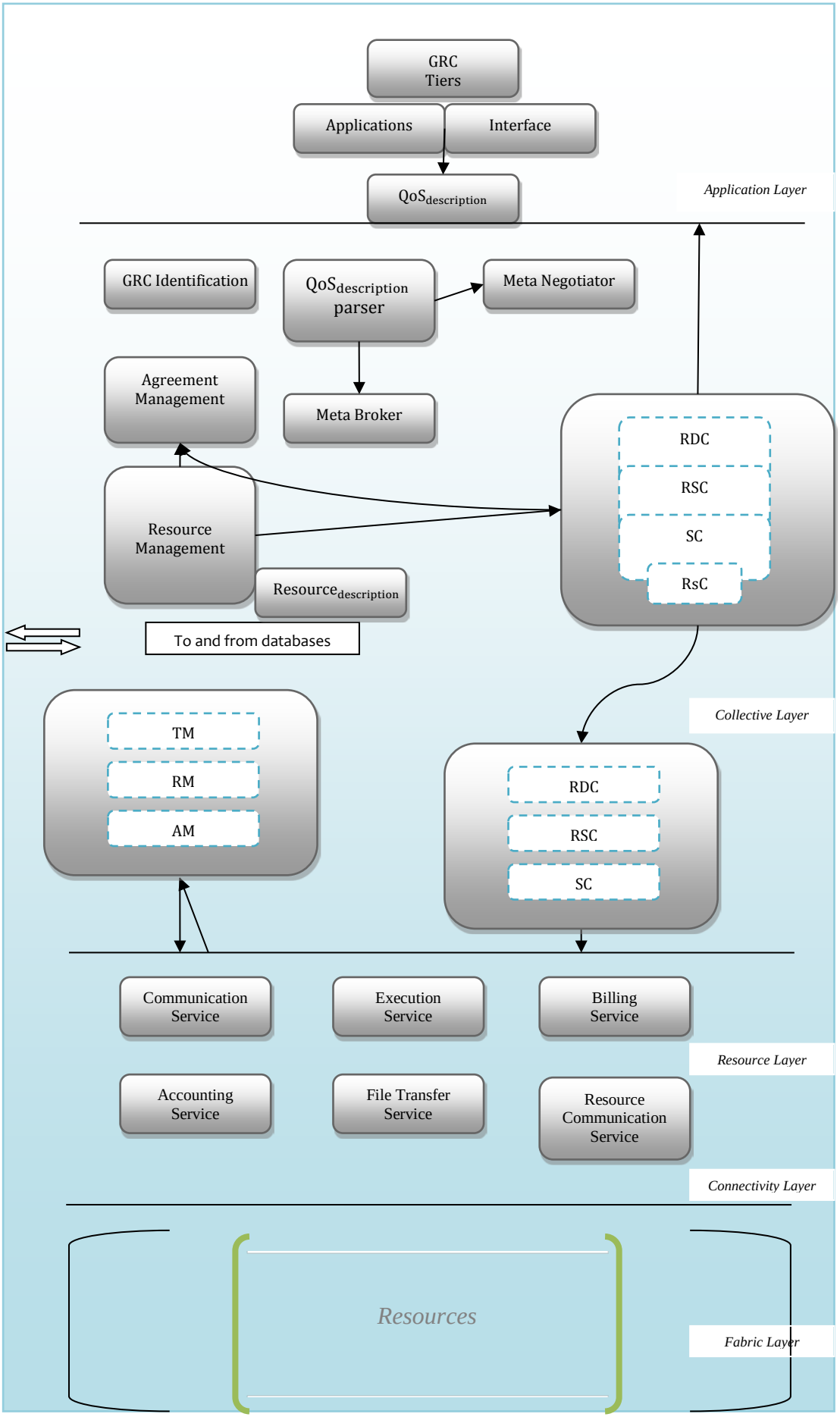


Figure 18: BGQoS Components

The combination of responsibilities by the different components of BGQoS are designed to carry out dynamic resource discovery, selection and allocation using the $QoS_{description}$ and $Resource_{description}$, as inputs. Resource information is stored in specific databases. The resources are typically heterogeneous, geographically distributed and operate under different policies. BGQoS operates on current information that is available and updated at regular time intervals, using a decentralised approach, where brokers can communicate with each other to provide a global solution meeting the GRCs requirements on-demand, as well as local scheduling capabilities.

Agreement maintenance is carried out via BGQoS. These agreements trigger reallocation and migration operations if necessary, using dedicated components for guaranteeing QoS promised to the GRC.

5.4. GRC Identification

The identification of the GRC, their tier and their administrative domain provides the authentication and authorisation specifications within BGQoS. Every GRC belongs to a tier in the multi-tier Grid user architecture. Each GRC must register and then a request is made to join a tier. Each tier provides the definitions that specify which resources the GRCs are authorised to use and what requests they can make. Moreover, each tier provides information on access rights in terms of local and global resources. Following is an explanation of the protocol followed for registering a GRC and assigning them an ID that corresponds to their tier:

- A GRC requests to register and join a tier.
- A GRC is assigned a tier:
  The expandable layered architecture allows each administrative domain to specify the authorisation levels for each tier. Each GRC within a tier is given the authorisation levels associated with that tier. The ID given to the GRC identifies locally the tier that the GRC belongs to, and since all global resource allocation occurs through the broker that is located within the same administrative domain, there is no compromise or misunderstanding in the authorisation levels acquired by each GRC.
- Each GRC signs on with their ID.
- The GRC is directed towards an interface that corresponds to their tier. Interfaces can be shared between different administrative domains or can be specific to each, specifying the operations each GRC tier is allowed to carry out. The interfaces

eliminate confusion and error while providing a mechanism for communication, feedback and task submission.

Once the GRC have been verified and submit their $QoS_{description}$ according to their authorisation level, they submit their request for execution. The request is acknowledged and the $QoS_{description}$ is sent to the $QoS_{description}$ parser.

5.5. $QoS_{description}$ parser

The responsibility of the $QoS_{description}$ parser is to extract the information from the $QoS_{description}$ submitted by the GRC. This information includes but is not limited to:

- Information relative to the GRC, such as the GRC_{ID} and location
- Number of tasks to be submitted
- QoS requirements
- Time constraint
- Cost constraint

Other information can be included in the $QoS_{description}$ such as usage requirements which must be met by the $Policy_{description}$ associated with the resources themselves. $Policy_{descriptions}$ are a part of the $Resource_{descriptions}$.

5.6. Meta-Negotiator

The responsibility of the Meta-Negotiator is to use the information extracted from the $QoS_{description}$ and liaise with the Meta-Broker on behalf of the GRC (Brandic et al 2008). The Meta-Negotiator feeds the Meta-Broker as an input and retrieves the output in return and feeds it back to the GRC.

5.7. Meta-Broker

The Meta-Broker is responsible for selecting an appropriate broker to carry out the resource operations on behalf of the GRC. The Meta-Broker uses broker ranking for the selection process by accessing a populated list of brokers and using specific ranking criteria to select the broker that fits the GRCs requirements best. The Meta-Broker serves as a method for higher utilisation of the Grid resource brokers and simplifying the broker selection process. Effectively a Meta-Broker provides a solution for the interoperability problem by providing GRCs with a uniform access point.

In an operational scenario, given a set of tasks that needs to be executed according to specific requirements. If the resource pool to be accessed is linked to different brokers, the meta-broker selects the appropriate broker using the resource information garnered from them, as well as other considerations, such as proximity or other criteria that could be specified. For this, a ranking method is used that assigns each Broker within a list of potential candidates with a rank value. If the selection criterion is based on priorities assigned to the broker, the description for this operation is presented in the following:

Input

$QoS_{description}$

BrokerList= { }; Potential Brokers to be ranked

ResourceList= { }; The Resources related to each of the potential Brokers based on GRPs connected and their $Resource_{descriptions}$

Priorities= { }; Broker priorities according to policies and agreements (Partnerships included).

Output:

Ranked Broker List

Start

(SizeOfList→ sl)

For i=1 to sl {

Compare Broker with Brokers above in the list

While (Broker Priority > higher Ranked Broker Priority)

Replace Rank value

Update Rank List

If Broker (Priority = higher Ranker Broker Priority)

{While resources available > resources available for higher Broker

Replace Rank Value

Update Rank List}

Else

Maintain list}

Return Ranked List

End

The $QoS_{description}$ submitted by the GRC and the $Resource_{descriptions}$ provided by GRPs are used as an input, as well as a list of brokers and their priorities. Brokers are ranked according to priorities and the resources related to them in the previous explanation. Other or additional criteria that may be a requirement for the GRC or the GRP can also be used. This returns a ranked list of brokers from the list of possible brokers.

Within BGQoS, resource information on available resources is current and updated at regular time intervals. This is taken into account in the resource broker ranking operation, in addition to the $QoS_{description}$ specifying the GRC's requirements. This process eliminates the possibility that a $Broker_i$ is ranked higher than $Broker_j$ even though $Broker_j$ can provide the resources immediately while the resources related to $Broker_i$ are tied up and are not available immediately.

5.8. Broker

A Broker is responsible for the resource related operations that are required to discover and select the appropriate resources using the information passed on from other components. Moreover, a broker holds and manages the communication between the GRCs and GRPs which is a necessary component for reaching an agreement. A Broker provides the interface through which task management can be initiated and the level of QoS maintained throughout the operation of the tasks with the help of the Monitor. A Broker is responsible for resource discovery, selection, scheduling, allocation and reallocation. Moreover, it is responsible for returning the results of completed tasks to the GRC.

5.8.1. GRC Commands

A Broker provides the operation required to return information to GRCs or execute their requests via one of the following commands:

Get Request Status

The GRC can request information on whether the resource discovery and selection process has been completed and whether sufficient resources have been allocated to meet their requirements. Three responses are possible:

- Request Granted: Returned if there are available resources that meet the GRCs requirements.

- Request Rejected: Returned if there are no available resources that meet the GRCs requirements && there are no resources that can meet the GRCs requirements (Figure 19).

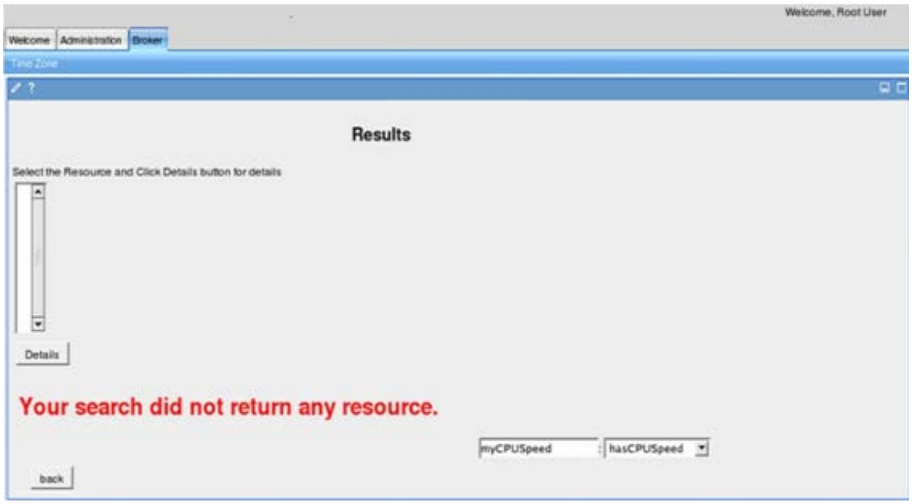


Figure 19: No Resources Returned

- Request Pending: Returned if there are resources that meet the GRCs requirements but are not currently available.

  Other commands are included in Table 6:

Table 6: GRC commands

Get resource List
Returns a list of the resources selected, as well as, detailed information related to them, such as the GRP and their location.
Get Estimated Execution Time
This command returns the Estimated Execution Time.
Get Estimated Execution Cost
This command returns the Estimated Execution Cost.
Get Task Status
Returns the status of the Task state, the task can be in one of the following states:

Task Pending
Task Scheduled
Task Queued
Task Running
Task Error
Task Completed
Task Failed.

5.8.2. The Resource Discovery Component (RDC)

The RDC is responsible for locating the appropriate resources that meet the GRCs requirements and produces a list of potential resources or resource sets according to the types of resources required in the $QoS_{description}$, the current state of resources and any other requirements that may be attached to the tasks submitted. The RDC uses the RR to retrieve information on available resources and uses this information to perform matchmaking between the submitted tasks and the resources according to the requirements of the GRC. The lists produced are not duplicated and are individually constructed.

The information stored in the RR contains both dynamic and static parameters relating to the resources that are available for selection. This information is updated at regular time intervals, which ensures that that information used is up-to-date and reflects the current state of the resources. The resource discovery component performs the first phase of the matchmaking process, the second being completed by the Resource Selection Component (RSC), introduced in section 5.10., which uses the information handed down by the RDC to produce a ranked list of potential resources. Following is a description of the operations of the RDC:

Input

$QoS_{description}$

$Resource_{type}$; Required resource types for executing tasks.

Number of Task; The number of submitted tasks

Output

List of potential resources

Start

While $_{av}R > 0$

{(Number of available resource in RR $\rightarrow_{av}$ R)

For each task = $task_t$ create a list P_t with length = R

{While $(_{av}R > 0)$

{For $task_t$, where t = 1 to R

get resource If { $resource_{description} = Type_t$ && ($resource_{description}$

= Perfect Match || $resource_{description}$ = OverQualified}

Add to list P_t } }

Return resourceSetLists P = {$P_1 \ldots P_R$}

End

where $_{av}R$ is the number of available resources.

5.8.3. The Resource Selection Component (RSC)

The RSC uses the output of the RDC which are handed down as a stack of resources or resource sets that can potentially meet the requirements specified by the GRC, to produce a ranked list of resources or sets of resources. The RSC is responsible for filtering potential resources according to the different criteria specified for the ranking process. Therefore, prior to carrying out the ranking operation, the ranking criteria must be retrieved by the RSC and used as input for its operation.

The criteria retrieved are then applied to the output handed down by the RDC to provide a ranked list of resources (in BGQoS, the top resource is given the rank "1"). Following is an explanation of the operation of the RSC:

Input

$QoS_{description}$;
ResourceSetList P; P = $\{P_1, \ldots . P_R\}$ retrieved from the RDC.
Ranking Criteria; The ranking criteria used, including ranking according to cost, time and partnerships. Other ranking criteria such as policies can be added.
Number of Task; The number of submitted tasks

Output

Ranked list of potential resources

Start

For $Task_t$, where t = 1 TO R
{Retreive P_t
Rank resources in P_t according to Ranking Criteria
Update P_t
Resouce with highest Rank in list RP_t}
Return RP
Store P

End

5.8.4. The Scheduling Component (SC)

The SC receives the task execution request and triggers the RDC and the RSC, requesting a set of resources that meet the requirements specified in the $QoS_{description}$ submitted by the GRC with the task execution request. Once the RDC and RSC complete their operation, as explained in the previous two sections, a ranked list of resource sets is returned and handed down to the scheduler. The scheduler uses the information handed down from the RSC to contact the resources, carry out task assignment, reservation operation and prepares the tasks for execution.

Two types of scheduling could occur, the first is that each task is dependent on the task before, i.e. must wait for the prior task to be executed, because it is dependent. The other type is a hybrid scheduling mechanism that carries out tasks both in parallel and in sequential manner. This can be explained that while some tasks are dependent on other tasks to be executed successfully, and therefore must be carried out sequentially, there may be other cases that are independent with resources available and therefore, could be carried out in parallel.

5.8.5. The Rescheduler Component (RC)

The RC is responsible for carrying out the scheduling operations while the application is executing, if triggered. This could occur if there is a violation of the agreement between the GRC and the GRP, or degradation in the level of QoS that is more than the ratio requested by the GRC. The information is returned to the GRC via the monitor explained in section 5.11. The rescheduler is only triggered if a specific set of conditions are met, as explained in the previous chapter.

Using the information returned by the RDC and the RSC, representing a ranked list of resources, produced initially for the first scheduling operation and saved in a database accessible by the RC, it goes through the sets to find the highest ranked available resource set by requesting up-to-date information. Once it locates a set of resources that is available, the RC carries out the scheduling operation for the remaining tasks. The description of the RCs operation is as follows:

Input

$QoS_{description}$

Ranked resource List RP: RP = $\{RP_1, \ldots., RP_R\}$

Number of tasks R-l: The number of tasks remaining to be rescheduled

Start

While there are remaining tasks:
{If Checkconditions = True;
{For $Task_t$
While there are resources available
Retrieve RP_t from RP
If (the next highest ranked RP_r is available)
{Assign $Task_t$ to RP_r
Send $Task_t$ to Q_{RP_r}}
Else
{Check the next highest}
If (no RP_r is available)
{Return $Task_t$ as failed }
}

End

5.9. Monitoring Component (MC)

The MC is responsible for overseeing the parameters associated with the execution process. The MC comprises multiple elements, each of which specifically performs specific monitoring responsibilities. These elements collectively oversee and monitor the different operations involved in successful task execution according to specific requirements and retrieve information on tasks and resources (Ropars et al 2006). The MC is also responsible for notifying the appropriate components of the information retrieved.

5.9.1. The Task Monitor (TM)

The task monitor is responsible for overseeing the tasks and collecting information on their location, execution, and level of QoS provided and the status of each task. The information retrieved by the task monitor in relation to each task can be categorised as:

- Task resource information
- Task status
- Task execution

5.9.2. The Resource Monitor (RM)

The RM is responsible for overseeing the selected resources. Following are the information retrieved by the RM in relation to each selected resource:

Expected Execution Start time for a Task

The expected start time is the time at which task execution is carried out, having completed resource related operations, including resource selection and scheduling and queuing. Moreover, this is the time when requirements have been transferred and resources prepared. It is when the Task Launcher initiates the actual execution of the task on the resource.

Resource Information

Resource information includes information on the following:

- Resource Load
- Resource Queue Length
- Resource QoS delivery
- Resource Characteristics

Resource Characteristics include both static and dynamic characteristics associated with each resource. Information on resources is updated at regular time intervals and the relevant information is retrieved by the monitor and is used in comparison operations to detect violations.

Resource Status

The resource can be in one of the following two states:

- Resource Functional
    - Resource Ready
    - Resource Busy
- Resource Failed

5.9.3. The Agreement Monitor (AM)

The AM is responsible for maintaining the agreement established between the GRC and the GRP. The AM uses the information returned by the TM and the RM to carry out the appropriate comparison operation and refers to the agreement to make sure that task execution is carried out according to the agreed terms between the parties. This includes that each task is submitted to the correct resource, that the level of QoS is in line with the agreement, and that the policies are adhered to, as well as other agreement parameters. If there is any violation to the expected execution scenario, the AM is responsible for returning the appropriate notification to the relevant parties and components.

5.10. The Resource Management Component (RMC)

The RMC is responsible for collecting resource information and storing it during the advertising phase. The RMC is also responsible for the RRs and for maintaining information on resources up to date using the available information at that specific time. The RMC is made up from the following elements:

5.10.1. The Resource Updater (RU)

The RU is responsible for retrieving resource information at specific time intervals and updating this information in the RRs. Resource information, initially stored in the RR, contains the advertised resource characteristics in the $\text{Resource}_{\text{description}}$. The RU performs the updating process, explained in the previous chapters , if i) the GRP requests an alteration ii) The information received by the RU differs from the current information stored in the RR relative to that specific resource identified by a specific $\text{Resource}_{\text{ID}}$.

5.10.2. The Resource Communicator (RC)

The RC is responsible for sending notifications to the GRP when an update operation occurs, sending them the newly updated resource information and the characteristics stored in the RR.

5.11. The Agreement Management Component (AMC)

The AMC is responsible for providing the tools for providing support for agreement enforcement, renegotiation and penalties if violations occur, as well as drafting new agreements if resource reallocation and migration take place.

5.12. Task Launcher (TL)

The TL is responsible for the actual execution of tasks utilising the services provided by selected and assigned resources. Once the scheduler submits the tasks, the responsibility is shifted to the TL for execution.

For every application, each task must be executed, required files accounted for, required input data accessible and each output collected. Required files must be downloaded and transferred if tasks are to be executed on resources that are not located in the same geographical location. Therefore, input/output operations are the responsibility of the TL. This includes validating that each task is receiving the correct input and that input is available, as well as, collecting the output from completed tasks.

5.12.1. The Local Task Launcher (LTL)

The LTL is responsible for handling all the tasks that are scheduled for execution on local resources. This element receives the execution requests, the scheduling details and initiates the actual execution of tasks using the information received.

5.12.2. The Global Task Launcher (GTL)

The GTP is responsible for handling all the tasks that are scheduled for execution on global resources, by synchronising the task execution operation between different sites, as well as carrying out global input/output operations.

5.13. The Task Migration Component (TMC)

The TMC is responsible for carrying out the required operations for tasks to be reallocated to other resources within the application run. The TMC is triggered if the conditions for reallocation and migration are met.

The TMC receives the new scheduling information from the RC which has been introduced above, and initiates the migration process. The TMC carries out the following operations:

- Synchronising the retrieval and placement of input data and input files.
- Task Launching on the new resources.
- Synchronisation between completed tasks on the first resource set and the migrated tasks on the new resources.
- Synchronising with the ABC, AMC and MC.
- The TMC may be triggered multiple times within the run of the application if the conditions for reallocation are met and the process is required.

5.14. The Accounting and Billing Management Component (ABC)

The ABC is responsible for calculating the actual cost and billing the GRC once the application has been completed.

5.14.1. The Accounting Manager (AM)

The AM is responsible for calculating the final cost for running the application, including migration costs, penalties and other criteria that must be added to the final figure. The AM receives the list of resources utilised, the price for utilising them per unit of time and the time they were utilised for. Moreover, the penalties, if any, in relation to these resources are added to the final calculation. The AM calculates the cost per GRP and sends the information to the BM which is responsible for collecting the payment from the GRC and returning it to the GRP.

5.14.2. The Billing Manager (BM)

The BM is responsible for payment and financial operations and communication between the GRC and the GRPs whose resources have been utilised. The BM receives the information required from the AM and forwards them to the GRC and the GRPs involved.

The BM uses the GRCs payment information, which is submitted as a part of the task execution request, to collect the required amount from the GRC and deliver the payment to the GRPs involved. Once the payment has been completed and received by the GRPs, the BM notifies both parties and their receipts are sent accordingly.

5.15. Summary

This chapter has introduced the operational components within BGQoS, their specification, responsibilities and the input/output operations related to each. Collectively, they represent a solid unit capable of carrying out the entire operational process covering different requirements, supporting GRC operations, GRP operation, resource operations, broker operations and task execution. Overall the components produce a model that is capable of carrying out QoS support while providing an expandable and flexible platform that could be deployed to support multiple domains within a diverse environment, supporting different types of users and resources. The next chapter is dedicated to BGQoS operations carried out using the implemented components implemented and the detailed explanation of each operation.

CHAPTER 6: BGQOS OPERATIONS

6.1. Introduction

In Chapter 4 the QoS model employed within BGQoS has been introduced, which was followed up in Chapter 5 with introducing the system components and design of BGQoS which allow it to support the QoS. This chapter complements the previous chapters, and provides further detail on the required operations of BGQoS.

6.2. Resource QoS Capabilities

A QoS GRC is capable of stating the $QoS_{parameters}$ they require. However, this raises the issue of determining whether resources can meet these parameters, and how to derive this relationship using the information available on the characteristics of each resource, both dynamic and static. BGQoS supports the allocation of resources that support a higher level than is initially required if the cost does not exceed the constraints set by the GRC. If we assume that an authorised GRC is requesting a computing resource in the shape of number of CPU cores, a computing resource in the shape of Memory in RAM and a storage resource, then one of three cases occurs within BGQoS. The first case is that of a perfect match, where the offer exactly matches the request in terms of type of resources and the level of QoS requested. The offer in this case if called a 'Perfect Match'.

Case 1: Perfect Match:

$$\text{QoS Parameters} = \text{Resource Characteristics}$$

The formula is as follows:

{(CPU = Requested CPU) AND (RAM = Requested RAM) AND (Storage = Requested Storage)}

The second case occurs when the level of service provided by a resource is higher than the level of service requested by the GRC. The offer in this case if called 'Over Qualified'.

Case 2: Over Qualified:

Resource Characteristics > QoS Parameters

The formula is as follows:

{(CPU ≥ Requested CPU) AND (RAM ≥ Requested RAM) AND (Storage ≥ Requested Storage) }

AND {(CPU > Requested CPU) OR (RAM >Requested RAM) OR (Storage > Requested Storage)}

In other words all three resource characteristics(CPU, RAM and Storage) must be at least equal to the requested QoS parameters and at least one of those three characteristics must exceed the matched requested QoS parameter (otherwise the case would be a perfect match rather than overqualified).

The third case occurs when the offer only partially meets the requested level of Service. The offer in this case if called 'Insufficient'.

Case 3: Insufficient:

Resource Characteristics < QoS Parameters

The formula is as follows:

{(CPU<Requested CPU) OR (RAM<Requested RAM) OR (Storage<Requested Storage)}

If either case 1 ("Perfect Match") or case 2 ("Over Qualified") is true in relation to a set of proposed resources, then these resources can be considered for allocation. The set of potential resources that meet GRC requirements, are called candidate resources

lists. BGQoS supports the negotiation process between the GRP and the GRC in this case. The model then filters the list for the most local and optimal solution meeting the GRC constraints using resource ranking.

In the third scenario where case 3 ("Insufficient") occurs, then the resources are deemed unfit and are not considered as potential resource sets that could be allocated, initially. Figure 20 illustrates the explanation above.

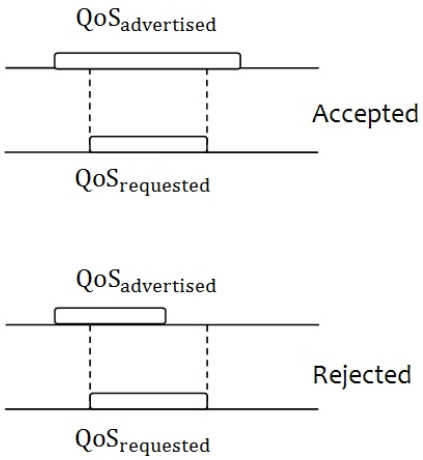


Figure 20: Accepted vs Rejected

The use of logical operators provides us with the option of introducing a variation of the BGQoS operational model where the GRC can specify a second requirement set in case the first one cannot be met. An OR operation is used in order to carry out this operation.

For the same request, a GRC may wish to provide two descriptions: CPU_i, RAM_i and $Storage_i$ as set of requirements (1) and CPU_h, RAM_h and $Storage_h$ as set of requirements number (2). The first set is called a main request and is given a priority value over the second set.

$$\{(\text{CPU} \geq \text{Requested CPU}_i) \,\&\&\, (\text{RAM} \geq \text{Requested RAM}_i) \,\&\&\, (\text{Storage} \geq \text{Requested Storage}_i)\}$$

OR

$$\{(\text{CPU} \geq \text{Requested CPU}_h) \,\&\&\, (\text{RAM} \geq \text{Requested RAM}_h) \,\&\&\, (\text{Storage} \geq \text{Requested Storage}_h)\}$$

If both requirements can be met, the main request is used.

6.3. Cost and Time Estimation

When the GRC submits a $\text{QoS}_{\text{description}}$, they submit the number of tasks to be executed. The completion of the matchmaking process means that a set of resources has been selected for the tasks to be carried out. Using the information submitted by the GRC and the information available on the selected resources, time estimation and cost estimation can be carried out. These estimations are used for the following:

- Provision of live information during the run of the application.
- Comparison between the delivered and expected level of QoS from the resources with tasks running.
- Meeting the Time and Cost Constraints.

6.3.1. Time Estimation

A service which supplies the GRC with an estimated time of completion once their tasks are submitted and a request is met by a set of candidate resources has been implemented within BGQoS. The model uses the information provided by the GRC in the request and the allocated resource characteristics to calculate an estimated completion time that is returned to the GRC. The estimated time can be calculated as the sum of the following time components, if the tasks are carried out sequentially:

$$\text{eT} = \sum_{n=1}^{k} (\text{eQT}_n + \text{eET}_n + \text{eTT}_n)$$

eT is the total estimated time for an application with k Tasks. eET_n is the estimated execution time of a Task T_n, eQT_n is the queuing time for a task T_n and eTT_n is the estimated file transfer time for the same task, T_n. This service is referred to as the time estimator.

However, if the tasks are carried out in parallel, then the estimated time can be defined as the time at which the final task will be completed $Time_{finish}$ and can be defined as:

$$Time_{finish} = \max_{n=1,k}(eQt_n + eET_n + eTT_n)$$

6.3.2. Cost Estimation

In addition to the time estimator explained, cost estimation can be requested by the GRC. The cost constraint specifies that a specific total cost should not be exceeded. Once a set of resources has been identified and selected, the cost of running the tasks could be calculated accordingly, using information on the price for using a resource per unit time p(t) and the time the resource is expected to be occupying the resource until completion, or the estimated τ. Since the resource usage cost in BGQoS is calculated based on a time basis, i.e. the price is per unit time and the GRC is charged for the period of time during which they use the resource.

If an application has k tasks:

$$eC = \sum_{n=1}^{k} p_n(e\tau_n)$$

Where eC is the predicted cost for the entire application and p (t) is the price of running task n on resource Ri for time t. eT_n is the estimated time it will require to complete task n on resource Ri. This service is called the cost estimator.

6.4. Phases of Execution

BGQoS phases of execution are:

6.4.1. Phase1: Information Retrieval

This phase includes two sub-phases:

- GRC requirements retrieval: the GRC request is parsed and their requirements are retrieved to be used in the next phase of the models operation. These requirements include the types of resources required, the QoS parameters required and the constraints that the GRC chooses to set.

- Resources information retrieval: The GRPs advertise their resources for local and global use if they wish to make them available for GRCs. Once the GRC requests have been received and the requirements information is retrieved by the model, the second part of this first phase is to retrieve the information on resources from the Resource Repositories (RRs) where up-to-date resource information is kept.

6.4.2. Phase 2: Matchmaking

A matchmaking decision, M_i, is defined as the decision that complies with:

- $QoS_{requested_i} \leq QoS_{expected_i}$, i.e. the resource selected meets the QoS requirement submitted by the GRC.
- $T_i > eT_i$, i.e. the estimated completion time must be earlier than the Time Constraint specified by the GRC.
- $C_i > eC_i$, i.e. the estimated completion cost must be less than the maximum Cost Constraint specified by the GRC.

Once the GRC requests are received and all the relevant information is retrieved, in addition to acquiring the information that is relevant on resources and their characteristics, the matchmaking phase is initiated.

6.4.3. Phase 3: Agreement

The negotiation process concludes with an agreement between the GRC and the GRP as mentioned above, the offer and conditions are included in an agreement which serves as a contract between the two sides. One of the main contributions of BGQoS is the simplification and automation of information retrieval from both GRCs about their QoS requirements and the information from GRPs about their resources, their characteristics, dynamic and static. BGQoS achieves this through using templates that both sides use for their respective purposes. The templates provide the model with the information required to map tasks to resources, by parsing the completed XML based templates, called descriptions.

6.4.4. Phase 4: Resource Allocation

Once the agreement is set, the GRC sends their tasks for execution and those tasks are allocated to the resource set that was agreed upon prior to the actual allocation. This is done via the task allocation component of BGQoS. Each task submitted is given a unique identifier $Task_{ID}$ when it is submitted to the resources that have been selected for its execution.

6.4.5. Phase 5: Monitoring and Maintaining Agreement

This is a very important phase in the operation of BGQoS as it is when the tasks are allocated to the resource sets; it has the responsibility to make sure that the GRC receives the level of QoS that was promised from the resources that were allocated. This monitoring process looks for any degradation in the parameters or any resource failures. If a violation is found, measures are taken to rectify them. These measures include migration, reallocation and penalty imposition.

6.4.6. Phase 6: Completion and Billing

Once all the tasks have been completed, the resources are released, the GRC is billed and the session is terminated. The GRP then decides whether to re-advertise the resources by making them available again. The dynamic information such as resource reliability (Dabrowski et al 2006) and availability are updated according to the latest information and statistics collected on the resource at that point.

6.5. Candidate Resource Accumulation

Candidate resources are discovered and ranked. Ranking candidate resources is accomplished via a multi-step filtering and ranking process that is initiated after accumulating the lists through matchmaking the $QoS_{descriptions}$ of the GRC with the $Resource_{descriptions}$.

If the level of QoS available in the $Resource_{descriptions}$ and the $QoS_{descriptions}$ from the GRC produce a result of "Perfect Match" or "Over Qualified" then the resource is added to the list of potential resources. To achieve this, two questions are asked:

- Is the GRC identified via the GRC_{ID} authorised to use the resources in the potential set? Do they have access?
- Is $QoS_{Available} \geq QoS_{Requested}$?

The answer to both questions must be a "Yes" for the set to be accepted as a potential set and added to the initial list.

6.5.1. Filtering: Meeting the Constraints

The first filtering process occurs at this stage. The potential resources that are identified are checked against the two constraints input by the GRC as part of their QoS description, the time constraint and the cost constraint.

For a resource set S_i containing resources $\{R_1, \ldots, R_n\}$ selected as a solution to a $QoS_{descriptions}$ for an application App_i containing n Tasks then S_i meets the constraints iff:

$$\sum_{s=1}^{n} pR_s(t) \leq C \;\&\&\; \max_{t=1,n} (time_{finish_t}) < T$$

Where $pR_s(t)$ is the price of R_s for the time it was allocated to the GRC, $time_{finish_t}$ is the time at which the final task is completed. C, is the Cost constraint and T, is the Time Constraint.

Once this is done, the lists of resource sets that do not meet the constraints set by the GRC are removed and the rest of the sets retained and included in a new list. This list is passed on to the next ranking stage.

6.6. Constraints Minimisation

There have been efforts in Web Services at exploring constraints and their role in service composition and service allocation (Aggarwal et al 2004, Guan et al 2006). In BGQoS we have defined an application as a collection of connected tasks. We have also modelled the Grid as a collection of variable types of distributed heterogeneous resources belonging to different owners that can be pooled together to execute the tasks that comprise an application. These resources can include, in BGQoS, computing and storage resources, so a Grid can include a set of resources $= \{R_1, R_2, R_{3,\ldots\ldots} R_i\}$.

The GRC may wish to select a constraint to be minimised if the option existed. For time minimisation: if we assume that there are more than one set s of potential resources that could execute n tasks submitted by a GRC according to their requirements and that $\max_{t=1,n}(\text{time}_{\text{finish}_s})$ is the expected completion time for final task on set s. If we consider the cost of executing the submitted tasks on resource s as c_s, then the aim is to choose the set of potential resources with the earliest $\max_{t=1,n}(\text{time}_{\text{finish}_s})$ while the following conditions are true:

$$c_s < C$$

And

$$\max_{t=1,n}(\text{time}_{\text{finish}_s}) < T$$

Where C is the cost constraint and T is the time constraint.

6.6.1. Rank According to the Proximity to $QoS_{description}$

The list from the previous step provided the sets of potential resources that are capable of meeting the requirements set by the GRC through the QoS description submitted initially. Next the resources contained in the list are compared with each other and stacked on top of each other where the top of the stack is the highest ranked set. The top set is ranked 1 and is reserved. If we assume that the requirements are as follows:

$$R > 80\%$$
$$C = 150 \text{ units}$$
$$T = 450 \text{ units}$$

And we assume the potential list of resource sets returned is as shown in Table 7.

Table 7: Potential Resource Sets

Set_{ID}	eC	eT	R
S_1	180.56	300.05	96.8
S_2	172.5	400.25	90.5
S_3	165.58	442.85	87.8
S_4	178.9	381.72	93.50
S_5	177.75	320.65	92.7
S_6	172.9	338.7	82.9

For the purposes of this example the GRC has chosen to minimise the cost while meeting the other requirements and constraints. In other words, the only time consideration would be that $eT < T$ without taking time minimisation into account. On the other hand, in terms of cost then $eC < C$ is taken with the objective of minimising the cost. In terms of reliability, it needs to be over 80%, $R > 80$. The sets returned meet these criteria and must be ranked, and the ranking is as shown in Table 8.

Table 8: Summary of swap operations for ranking

Sets	C_{diff}	eT_{diff}	R_{diff}	*Swap Ranks*
S_1, S_2	-8.06	100.2	-6.3	Yes
S_2, S_3	-6.92	42.6	-2.7	Yes
S_3, S_4	13.32	-61.13	5.7	No
S_3, S_5	12.17	-122.2	4.9	No
S_3, S_6	7.32	-104.15	-4.9	No

The minus signs in the table above represent the differences for the second set in the comparison. Therefore, S_3 is selected as the highest ranked resource set solution. Figure 21 illustrates the resource operations including time and cost estimation.

6.6.2. Combination Ranking

A ranking process combining the two ranking steps above is available within BGQoS. Where cost or time minimisation is carried out and the top f sets ranked $S_1 \rightarrow S_f$ are rearranged according to the proximity to $QoS_{Requested}$.

Combining the two ranking steps provides a more reliable, accurate and relevant selection method, however, it does incur more overhead for the resource selection phase. Moreover, the minimisation preference of the GRC must be included in the $QoS_{description}$.

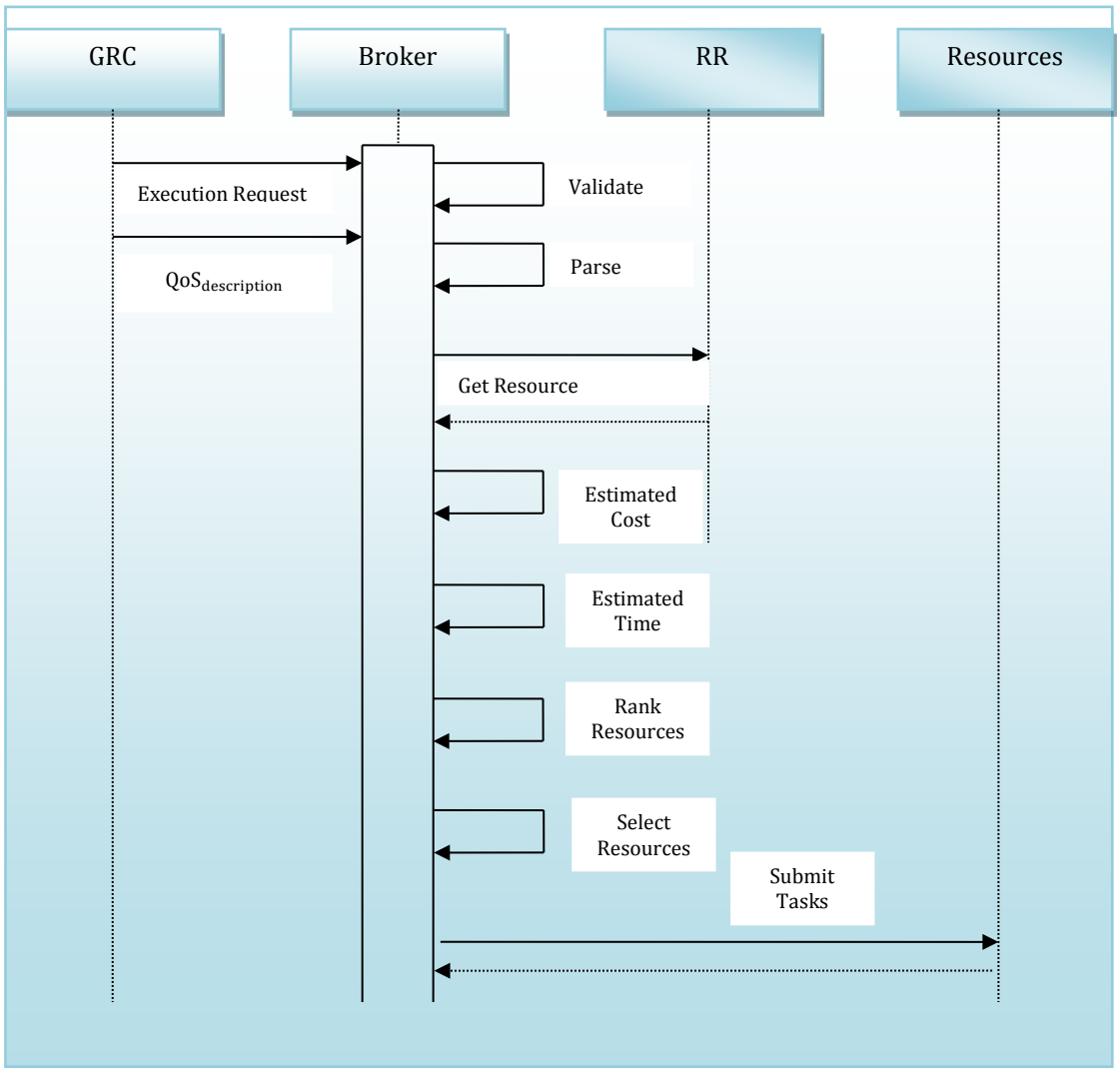


Figure 21: The resource operation process

6.7. Policies

Policy matchmaking (In et al 2004) is supported by BGQoS. It performs the required operations in policy management, allowing GRPs to specify the policies for their resources, and submit a $policy_{description}$ explaining the specific requirements. As explained throughout this chapter, BGQoS receives an execution request and a $QoS_{desciption}$ at the start of a session. Using the information available on resources, static and dynamic, BGQoS selects the sets of potential resources before filtering the lists and ranking them. Even though, the resources selected are capable of carrying out the tasks according to the $QoS_{desciption}$, the usage policies set by the GRP must be taken into consideration. For this another filtering step is proposed, based on the resource usage policies attached to the resources. For example, the GRC may require that they have access to the temporary files produced while the tasks are executing on

the resources they acquire. In this case, if two sets of resources are identical, with the difference that the temporary files are saved for one set and they are deleted for the other, the first set is the choice set. Whether the temporary files are kept or deleted is specified in the usage policies submitted by the GRPs.

A $Policy_{description}$ can contain multiple types of information. It may include:

- Local policies in relation to the resources themselves. For examples, if a GRP is advertising resources with a specific amount of computing power, it must also specify the percentage of the resources it is willing to contribute within a specific period of time.
- Indication whether the resources are allowed to be allocated to global GRCs or are restricted to local GRC usage.
- The conditions of cooperation between different GRPs for providing a combined resource set that meet the GRCs requirements.
- The operational restrictions for using their resources, such as the maximum load or the maximum reservation period in relation to specific GRCs.

Each policy is attached to the $Resource_{ID}$ and the GRP_{ID} and is stored with the resource it is associated with in the RR. Like the resources themselves, the policies are accessible for authorised entities, allowing those entities to carry out retrieve, update/edit and delete operations when necessary. Moreover, the policy itself is produced in an XML based format. This allows BGQoS to parse through the documents and extract the required information. BGQoS uses the extracted information to locate the resources that are attached to usage policies that meet the GRCs requirements and selects the appropriate resources accordingly.

6.8. Matchmaking

Matchmaking is the core operation of BGQoS. A flexible, accurate, simple and expandable approach to matchmaking based on the $QoS_{descriptions}$ submitted by GRC is implemented. These $QoS_{descriptions}$ outline the high-level requirements for their tasks. Moreover, as explained in previous sections, descriptions are submitted in accordance to policies defined within a multi-tier user grouping model. The environment for which BGQoS is proposed itself includes multiple domains and many applications that differ in terms of their requirements and the way they may use the resources provided by different Grids. It is therefore very important that BGQoS is

easily expandable to these variable domains. It is at this stage of active resource discovery and resource selection and matchmaking that resources through their characteristics and the QoS descriptions where BGQoS provides the basic functionality that can be built upon via other components to tailor to the different domains.

6.8.1. Multi-Tier Interface

In BGQoS a multi-interface model is implemented corresponding to the multi-tier GRC architecture introduced. It also provides the GRC with the ability to specify high-level QoS requirements that form the core of the $QoS_{description}$, since we are aiming to separate the GRC from dealing with the low level Grid infrastructure, directly. They also serve as the entry point to BGQoS and session initiators. Every GRC logs into an interface that is specifically tailored to them. This is in line with our multi-tier user model. This simplifies validating the GRCs authorisation for making their requirements as well as simplifying requirement setting for the GRC. Moreover, it simplifies requirement extraction for the relevant components that are concerned with searching for the appropriate resources.

The design and the components of the interface might differ according to domain or specific administrative requirements. Every GRC in our model belongs to one specific tier. This user model can be expanded to accommodate as many tiers as every administrative domain sees necessary. Different interfaces have been developed for each tier with the objective of allowing each GRC access to the resources to which they have permission, as well as, allowing each GRC to specify their QoS requirements and constraints as specified in the definition of their level.

The User interface component of BGQoS serves the following functions:

- Provides the entry point to the scheduling process, receiving GRC execution requests and $QoS_{description}$
- Limits each GRC to their level of privileges reducing errors and validating requirements.
- Receives GRC Commands, such as requests for Task Status information and level of QoS delivered.
- Invokes the proper elements of the model to commence executing tasks.
- Receives the output from successfully completed tasks and applications.

Once the registered GRC logs into their correct interface, they are capable of specifying the QoS they require from the resources they require.

6.8.2. GRC Request and $QoS_{description}$

The matchmaking process is initiated *via* a request by the GRC containing an application execution request and a description of the requirements in terms of QoS parameters, constraints and other relevant information that is specific to each domain. The GRC must also submit the number of tasks that are to be sent for execution. The importance of this information is explained in the following sections of this chapter. These documents are parsed and the descriptions are extracted. This information is used in the next step.

6.8.3. Resource Discovery

Potential resources are accumulated via the $QoS_{description}$, the $resource_{description}$ and using the components and operations introduced in this and previous chapters. The potential list of resources and resource sets is ranked in the next resource operation.

6.8.4. Resource Selection

The resource selection phase within BGQoS is an execution phase in its own right within BGQoS as explained in previous sections, unlike previous approaches where resource discovery and selection were combined within one execution step. Once the resources are discovered in the previous step, the ranking process begins.

Resource ranking is achieved through following the resource ranking options:

- Resource Ranking according to GRC request.
- Resource Ranking with Cost Minimisation
- Resource Ranking with Time Minimisation
- Combinational resource Ranking.

6.8.5. Scheduling and Executing Tasks

Once resource ranking is completed, the resource set S_i with rank $= 1$, the top of the resource set stack, is selected. Once the resource set is selected, the resources are reserved and tasks are sent to those resources for execution according to the schedule produced in relation to the selected resources.

6.9. Partner and Global Access to Resources through Brokers

In Grid environments, Grid users must rely on resource brokers to discover the appropriate resources that meet their requirements, which has led to this surge in the development of different resource brokers after the initial efforts of the Globus Grid Resource Broker (Pathak et al 2005). However, these different resource brokers have no clear way of communicating between each other, or a clear protocol for communication between a GRC and a GRP through a set of resource brokers. Moreover, in BGQoS the top level GRCs are allowed access to different resources on different Grids, which presents the question: Which Grid should I use?

In BGQoS once GRCs submit their execution requests and $QoS_{descriptions}$, they expect that the appropriate resources be located and selected. In addition to the methods that have been introduced to achieve this goal, a component has been implemented in BGQoS, which operates as a high-level communication platform between different Grids and deciding which Grids to communicate with in case local resources cannot meet the GRCs requirements.

The first step is to populate a list at each location with a list of potential brokers that can be communicated with. In the following step the ranking technique is tailor implemented and applied by BGQoS, by implementing broker ranking that follows the following criteria:

- Priority: The first filtering phase is based on whether mutual agreements exist between different, Grids, organisations and domains. If these agreements exist, only brokers included in these agreements are considered and the rest are discarded. The newly populated list of potential brokers proceeds to the next filtering step.

- Location: The proximity of the location of the broker is important and taken into consideration when selecting the right broker, in keeping with BGQoS's objective to shorten distances between GRCs and the resources they require, for more efficient and smoother allocation.

- The resources available and the number of successful tasks that have finished successfully over a period of time, which can be calculated using up-to-date information.

- The load of the Grid that the broker is a part of.

$$GB_i \xrightarrow{rank} 1$$
$$GB_n \xrightarrow{rank} n$$

If the geographical location or distance is the criteria followed, then the process is shown in Figure 22:

Step 1: **Partner Brokers**

Contact the Broker Repository and get a set of Partner Brokers, PBrokerSet

For all Brokers pj $\in$ PBrokerSet, calculate d(pi, pj), where d is the distance

Select p_{min} with minimum distance to pi

Contact partner broker

$S(i) \leftarrow S(p_{min})$

Repeat for all Brokers in PBrokerSet if resources do not meet QoS requirements

Step 2: **Global Brokers**

Contact the Broker Register and get a set of Global Brokers, GBrokerSet

For all Brokers gj $\in$ GBrokerSet, calculate d(gi, gj), where d is distance

Select g_{min} with minimum distance to gi

Contact global broker

Repeat for all Brokers in GBrokerSet if resources do not meet QoS requirements

Figure 22: Broker ranking according to distance

However, just like resource ranking, it is possible to introduce ranking criteria according to different requirements for different domains, organisations or agreements.

6.10. Reallocation

In the previous sections BGQoS' aim at assigning the right tasks to the right resources by a matchmaking process driven by $QoS_{description}$ submitted by GRCs and using available up-to-date information on resource characteristics and QoS levels has been explained. Sustaining this level of QoS is a major objective of BGQoS. The combination of appropriate resource selection and the sustainability of the level of QoS provided by the selected resources provides the guarantee to the GRC that their $QoS_{Requested}$ is met and maintained until the completion of the tasks submitted. The premise that both parties will adhere to what they agree upon is documented in a contract that is initiated by the GRC, received by the BGQoS and offered by the GRP. If there is a violation of the contract, which might occur for multiple reasons, including resource failure and performance degradation, then the reallocation components in BGQoS are activated.

6.10.1. Issues to Consider

The decision to reallocate, from the GRCs point of view must be put into the context of whether it is beneficial or not. There are a number of issues to consider:

- The percentage of tasks that have already been completed.
- Whether there are available resources that could be allocated immediately while still meeting the GRC requirements.
- Whether the total cost, including the cost of moving the tasks from one resource to another is viable and within the constraints.

If the conditions are met, then reallocation is viable and beneficial, the rescheduling and reallocation components of the BGQoS are triggered. Our model performs rescheduling and reallocation in two ways; the first is to a migrate to a different set of resources for guaranteed QoS GRCs, and, the other performs resource swapping for best effort GRCs, if there are resources available.

6.10.2. Reallocation for Guaranteed QoS GRCs

The first approach that BGQoS uses for Guaranteed QoS GRCs, migrates the application to an alternate list of resources that meet the GRCs requirements. This approach is implemented using an improved stop/start approach to rescheduling, using the ranking mechanism introduced above, in which multiple lists of resource sets are ranked according to specific criteria.

Stop/start is a rescheduling mechanism that halts the application at a specifically defined point in operation and performs migration to another list of resources that are available. When the running application encounters a contract violation, reallocation is initiated and the tasks are to be migrated. When this occurs, the tasks are stopped, user specific data is check-pointed and the application is terminated. The application is restarted on the second list of resources that is available, using the check-pointed data.

6.10.2.1. Reallocation via Ranked Lists

BGQoS accumulates information on resources and identifies a list of candidate resources that could potentially meet the GRCs requirements and stay within their constraints. Resources are ranked according to different criteria, including better matching and constraint reduction through multiple filtering processes. The ranked lists, are stored until the tasks are carried out successfully, the application is completed and the results are returned to the GRC. The reallocation process within our model uses this information for resources migration when is required.

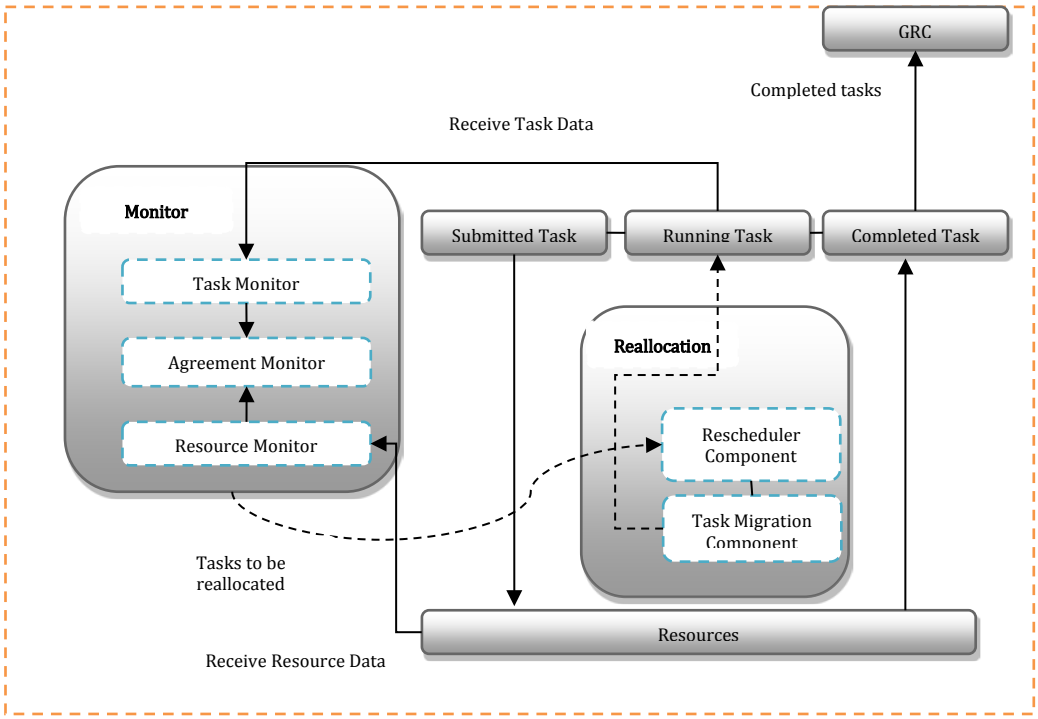


Figure 23: Monitoring and Reallocation of Tasks

6.10.3. Tolerance Ratio

It is necessary to specify the points at which reallocation is viable, numerically. For this a QoS ratio is introduced, calculated at specific time intervals, as the ratio between the values of the expected QoS to be delivered at time i, $QoS_{expected_i}$ and the actual QoS delivered at that point, $QoS_{delivered_i}$, effectively calculating the percentage of QoS delivered in relation to the original requested and agreed upon QoS. A Tolerance Ratio TR is specified by the GRC, expressing the percentage to be tolerated if the $QoS_{delivered_i} < QoS_{expected_i}$. If none is set by the GRC, a default value is referenced. If $QoS_{delivered_i} < QoS_{expected_i}$ then the actual Delivered Ratio DR as a percentage is calculated as:

$$DR = \frac{QoS_{delivered_i}}{QoS_{expected_i}} \times 100$$

Case 1:

If DR $\geq$ TR no migration is necessary and the application continues its operation normally until the next specified time for calculating a new DR, where the process is repeated.

Case 2:

If DR $< TR$, the rescheduling and reallocation components of the BGQoS check how much of the application has been completed. This process is directly related to the second of the issues to consider, specified earlier in this section, relating to whether there is any benefit in migrating the application at that specific point in its execution cycle.

$$\text{Migration decision: } \begin{cases} \text{Migration, if DR} < TR, \\ \text{No Migration , if DR} \geq \text{TR} \end{cases}$$

At this point, there are a number of issues to consider, including:

- The size of an application
- The QoS requirements
- The Cost constraints specified by the GRC
- The Time constraints specified by the GRC

The first issue is solved by proposing a percentage of tasks completed, this both eliminates the size of the application as parameter as well as maintaining the GRCs control over the reallocation procedure, this parameter is called Completion Ratio *CR*. The percentage of actual tasks completed can be calculated as Actual Completion Ratio ACR:

$$\text{Actual Completion Ratio} = \frac{\text{Number of completed tasks}}{\text{Total number of tasks}} \times 100$$

If $ACR \geq CR$ then no migration is carried out and the penalties are incurred, if $ACR < AR$ then the third issue is to be considered. In general:

$$\text{Migration decision: } \begin{cases} \text{Migrate, if DR} < TR \text{ and } ACR < AR \\ \text{No Migration , if RDR} \geq \text{TR or ACR} \geq \text{CR} \end{cases}$$

The third issue is whether the cost of migration is within the constraints of the GRC. If the first two conditions are met in (1) and (2) then the cost of migrating the resources is calculated as Migration Cost ($C_{migration}$):

$$\begin{aligned} \text{C}_{\text{migration}} = (&\text{Cost (set}_{1_i}) + \text{Cost (Migration)} \\ &+ \text{eC}_{\text{set}_{2_j}}) - \text{Penalties imposed on GRP providing set}_1 \end{aligned}$$

The condition to be met is:

$$\text{C}_{\text{migration}} < C$$

The Time constraint is the final issue to be considered. While the resources lists have fulfilled the initial Time Constraint requirements, the migration time must be added to the total execution time, introducing a newly calculated estimated time of Completion, $\text{T}_{\text{migration}}$:

$$\text{T}_{\text{total}} = \text{T}_{\text{set}_1} + \text{eT}_{\text{set}_{2_j}} + \text{Time required for Migration operation.}$$

The condition to be met is:

$$\text{T}_{\text{total}} < T$$

If the conditions introduced in this chapter are met, then migration is initiated. The migration operation specifies that under specific conditions, which are presented above, it is possible that tasks may migrate to a different resource before the final execution result is returned to the GRC.

6.11. Reallocation for BE GRCs

A simple resource swapping reallocation process is introduced for Best Effort GRCs. If a resource fails, then the model checks if there are any other resources that are available, locally. If there are, then a reallocation process is initiated. If there are not, the tasks are returned to the GRC and they are informed that the application has failed. In this case the application must be resubmitted and restarted.

6.12. Summary

This chapter has described the operations undertaken within BGQoS and its components in order to successfully complete the responsibilities that collectively cover the entire sequence of processes that are supported. $QoS_{description}$ driven resource discovery and selection operations are explained within this chapter, in addition to the states of the response generated when resource information is queried and whether it has the potential to be a candidate resource. Resource ranking and filtering operations have been introduced in parallel with cost and time minimisation capabilities. Moreover, broker selection and ranking operations have been explained. Reallocation operations have been explained in this chapter and the conditions that need to be met in order for them to be carried out. The tolerance ratio serves as the minimum level of QoS that a GRC is willing to accept from the resources selected. The next chapter is dedicated to the simulation tools that have been used and developed in order to evaluate BGQoS.

CHAPTER 7: SIMULATION

7.1. Introduction

This chapter explains the motivation for using a simulated environment to implement and evaluate BGQoS. This chapter is split into three parts. The first part elaborates on the reasons behind the motivation and current simulation tools available. The second part explains the simulation tool that has been chosen in detail and presents its relevant components. The third part explains the alterations and expansions carried out on the toolkit in order to accommodate the requirements and the components of BGQoS.

7.2. Motivation for Simulation

Simulation in Grid Computing is necessary due to the difficulties in locating available Grid test-beds, the price of using these test-beds if they are available, the limited number of these test-beds and administrative complexities of using them, as well as other technical, logistical and monetary reasons. The simulation solution is the logical substitute. Moreover, simulation allows multiple runs, and experiments to be repeated for better results, accurate analysis and concrete development. This makes simulation not only logical but also necessary and practical for testing new Grid models. However, current Grid simulation tools are limited and do not provide the user with the ability to completely simulate real-life environments, resources, applications and users. Issues such as: negotiation between the GRC and GRP; contractual agreements between GRCs and GRPs, and, multi-application, multi parameter guaranteed QoS specific support, are some application-related issues that can only be partially simulated using current simulation tools.

For applications that aim to use the Grid Computing infrastructure as a vessel to carry out their operations, the functions mentioned above are important and essential to introducing more multi-domain applications into the Grid computing mesh of heterogeneous resources. All of the above have led to our decision to use simulation as part of our methodology for implementing and testing our model.

However, because of the many components of BGQoS, the need to simulate a real life environment, present more accurate results and carry out relevant experiments, it was necessary to expand one of the current simulation tools to accommodate BGQoS requirements. The tool that has been chosen is the GridSim toolkit (CLOUDS Lab 2010). GridSim is flexible and well documented; however, it lacks some of the functionalities mentioned above. These functionalities are required to make

simulating BGQoS possible. The decision to use GridSim was down to its ability to support many more functionalities than other simulation tools. Not only is it, in our opinion, the most comprehensive, but it was designed in a way which allows additions and alterations to be made. Moreover, GridSim's layered architectural model makes it easy to understand and add to. (CLOUDS Lab 2010, Sulistio et al 2007, Buyya et al 2002).

7.3. Current Simulation Tools

This section introduces the simulation tools and technologies that are available and could be used to simulate Grid environments.

7.3.1. OptorSim

OptorSim (DataGRID 2004), Figure 24, was developed by the DataGRID (2004) initially to test their algorithms; it is available as open source software for Grid users. OptorSim takes a Grid configuration file as an input. Configuration files define: the resources; Grid topology; Tasks; Associated files, and, one of the parameters of the algorithm to be used. This is followed by choosing one of two types of optimisation algorithm:

- Scheduling algorithms
- Replication algorithms

OptorSim also allows the user to visualize the performance of a specific algorithm. It provides a set of measurements which can be used to quantify the effectiveness of the optimisation strategy under consideration, hence focusing on optimisation and data replication.

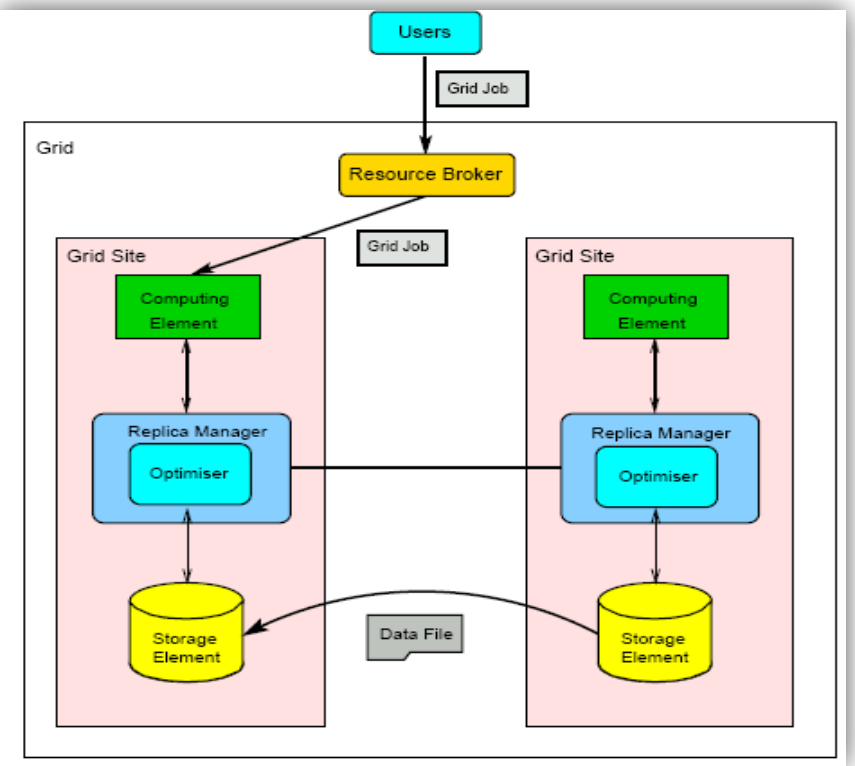


Figure 24: OptorSim Architecture

7.3.2. SimGrid

SimGrid (1999), Figure 25, is a toolkit that is implemented in C programming language and it was created at the University of California, San Diego (UCSD). SimGrid provides core abstractions and functionalities that could be used to simulate specific distributed computing environments. Specifically, the aim of SimGrid is to provide the tool for carrying out research in resource scheduling in distributed environments.

The main step in a SimGrid simulation is the creation of resources, which are assumed to have two performance parameters, latency and service rates. These parameters are used to simulate performance using a vector of time-stamped values or constants.

SimGrid V2 introduced in 2003 introduced a new layer. This layer provided the toolkit with the capability to build simulations in terms of communication agents alongside its basic capability of scheduling tasks on resources (Legrand et al 2003, Casanova et al 2008).

In 2006, a model called Grid Reality and Simulation (GRAS) was deployed on top of SimGrid V2 in order to facilitate the operation of simulated codes in real time environments. This model was built on top of the new software layer of V2; the Meta-SimGrid (MSG) in simulation mode and is built on top of the socket layer in real mode, introducing what is known as SimGrid V3 (Casanova et al 2008).

Figure 25: SimGrid Architecture (Casanova et al 2008)

The main disadvantage of SimGrid is because of its restriction to a single scheduling entity and time shared system, it is difficult to perform simulations of multiple users, resources or applications, each with separate policies and specifications.

7.3.3. MicroGrid

MicroGrid (2004), Figure 26, is an online simulation tool that was developed in the University of California in San Diego (UCSD). MicroGrid is modelled for the Globus toolkit and allows applications created in Globus to be carried out in a controlled emulated environment.

The main aim of MicroGrid is to provide an online platform that supports the simulated execution of real life applications. One advantage of MicroGrid is that it supports running applications that use dynamic resource allocations. Moreover, it provides a vessel for repeatable experiments in order to observe and study design aspects for applications and middleware, exploration of extreme circumstances and choices of application deployment, Grid resource allocation and network design.

Figure 26: MicroGrid infrastructure (MicroGrid)

MicroGrid reads a virtual Grid configuration file, and then uses the configuration to build corresponding simulation objects required to create the virtual Grid. These simulation objects include network elements and computing resources. MicroGrid allows applications and middleware to be executed on virtual machines, allowing this execution to be carried out near real-time. The user of MicroGrid specifies a set of virtual resources before specifying the physical resources to be used for the compute and online network simulation. The user will then be able to submit the application as a task on the virtual Grid, and observe the execution (MicroGrid, Huaxia Xia 1999, Xin Liu 2004, Richard Huang 2006).

The main disadvantage of MicroGrid is that applications need to be developed using the Globus toolkit which produces a significant amount of overhead. Moreover, modelling a large number of applications, environment and scenarios could require a significant amount of time.

Other simulation tools are available, such as Bricks (2002) , GangSim (2006) and Grid Scheduling Simulator (GSSIM) (2009).

7.4. GridSim

GridSim is a comprehensive, general purpose toolkit for simulating heterogeneous resources, users and applications. Resources can be single processors, multi-processors or distributed memory machines. It can also be used for simulating different administrative domains, which in turn supports the simulation of multiple policies, schedulers and organisations in a distributed computing environment. All of the above are elements in designing a simulated Grid environment.

The GridSim toolkit currently provides the most comprehensive package that could be used for simulating resources, applications, users, network connecting devices and organisational layouts. It also supports the composition of user-centric applications, simple resource discovery and simple resource management. Most important, this allows different scheduling algorithms to be simulated and evaluated, which is the reason why GridSim is preferred and used by many researchers.

Nimrod/G (Nimrod/G 2010)(Buyya et al 2000) has been used by the laboratory that has created GridSim and its developers as the standard resource broker for the evaluation of cost and budget constrained scheduling algorithms. These algorithms are with time, cost and time/cost and conservative time optimisation, as introduced in the previous chapter.

However, GridSim only provides the base and simple operations that are unable to fully and accurately simulate a true Grid environment, the users, and schedulers. Moreover, the definitions and creative flexibility for users, tasks and resources is limited to what is standard. In the package, users are initially created and immediately are required to create all their Gridlets or objects that represent real tasks, and are to be simulated. More formally each user is called a *user entity.* It is worth noting that in real Grid environments, this is not the case, as users are free to and should be able to create their tasks when they choose at any point.

Because Nimrod/G is a user-centric scheduler, when the user sends his/her tasks to be scheduled and requests the resources to do so, the resource broker that is connected to this user tries to satisfy the users request without taking into consideration any of the other requests from any of the other users. This greedy method is unsustainable; it does not take load balancing into consideration and neither does it consider congestion in the Grid. As mentioned before, Nimrod – G (Nimrod/G 2010) provides cost, time and cost/time optimisation scheduling only,

however, the reality is that there many other scheduling policies and models that could be implemented and are in most cases, application specific.

7.4.1. GridSim Features

The following lists the basic features that are provided in the GridSim toolkit:

1. It allows the modelling of resources. These resources are limited to PCs, workstations and clusters. Resource are of two types:

   - Time-share resources: A single computational entity or processor is able to execute more than one task at any given time, using a round robin approach. In this approach, each task is given a share of the processing power.
   - Space-share: A single computational entity or processor can only execute one task at any given time. It must therefore complete the execution of any current task allocated to it before it can start another task.

   Resources contain discrete machines, the number of which is decided by the simulator with no upper limit. Each machine contains a number of *Processing Elements (PE)* representing processors or CPUs. The processing power of each PE is calculated by the standard measurement in Grids, millions of instructions per second (MIPS). Resources can be allocated time zones, with weekday, weekend and holiday options also available. This gives the resource a local time and allows the modelling of workloads accordingly. Resources can be reserved in advance.

2. Different types of applications can be simulated. Different parallel application models can be simulated.

3. Applications are made of a collection of tasks; these tasks could be specified by the user to not be of the same category. This allows for a level of heterogeneity, allowing the user to specify the number of compute-intensive tasks and the number of data-intensive tasks that could be simulated as part of the same application.

4. There is no maximum number of tasks or upper limit to how many should be compute-intensive or data-intensive.

5. Multiple users could be created with different user related properties.

6. Multiple users can submit their tasks to the same resources simultaneously. This allows the implementation of scheduling techniques that allow competitive resource allocation between users. This competitive environment is essential to the realistic simulation of Grid resources.

7. Network specification:

- GridSim allows the planning, design and simulation of an entire network.
- Network speeds can be specified.
- Networks can be linked to users, resources, schedulers and other networks.

8. Simulation of dynamic schedulers is supported, in addition to static schedulers.

9. Statistics and information on selected operations can be recorded and used for analysis.

7.4.2. GridSim Architecture

The layered architecture of GridSim is shown in Figure 27. Each one of the layers provides an interface to the layer on top of it. The bottom layer, also referred to as the first layer is concerned with the Java runtime environment and the JVM (Java Virtual Machine). Their implementation is available for both Single Processor System (SPS) and Multi-Processor System (MPS). The second layer contains the simulation package, SimJava2, which is a basic, discrete-event simulation package. A release of this package has recently become available. The third layer contains the GridSim toolkit. Modelling the simulation, resource allocation, recording stats, Grid Information Services and other parts of the toolkit are the main concern of this layer. The fourth layer of the architecture is concerned with actual simulation of resource brokers and schedulers. The final upper layer or fifth layer is where the modelling of applications and resources for different scenarios defined by the users are implemented. This upper layer uses the services of the layers below it.

SimJava2 is, as explained above, a discrete-event simulation package written in Java. SimJava2 simulations contain a number of interacting *entities*, which are used by GridSim. An *entity* is the simulated component that interacts with other components in SimJava2. Each one of these entities runs in its own thread. Entities are represented by the class Sim_entity.

Sim_entity contains all the functionalities that are available for the entities in the simulation. A subclass of Sim_entity must be created to define an entity type. The body

() of this subclass contains the required behavioural characteristics of each entity, which must be overridden in the subclass. Entities could be: users; resources; network devices; Grid Information Services, and, statistical recorders. Every single entity is an instance of a Sim_entity subclass.

A more detailed explanation of the GridSim entities is in the following section of this chapter.

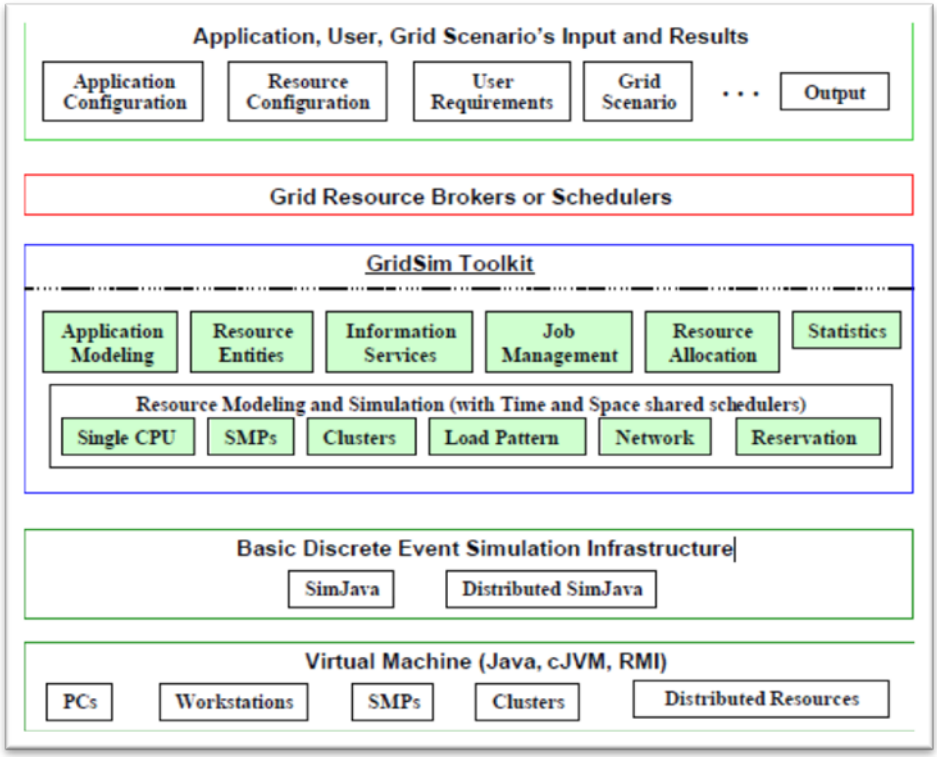


Figure 27: GridSim layered Architecture

7.4.3. Entities

This section explores some of the entities in GridSim, what they represent and their roles.

7.4.3.1. User

Every Grid user to be simulated is represented by a User; each User is represented by an instance of the User entity. Each User is distinguished from other Users via the following properties:

- Number of tasks to be submitted.
- Execution time of each task
- Scheduling optimisation strategy, one of the following:
    - Time
    - Cost
    - Cost/Time
- Task creation rate, which also defines the level of User activity.
- Time Zone.
- Deadline, Budget or Deadline and Budget combined.

N_j users can be created, competing for a common resource type *j*. Each Grid User has *tasks* to execute on a resource *r*.

7.4.3.2. Resource Broker

Each User is connected to a resource broker; each resource broker is represented by a Resource Broker entity. Each user submits their tasks to the resource broker they are connected to, and the resource broker sends the tasks to the resources according to the Users optimisation strategy: Cost, Time or Cost/Time.

7.4.3.3. Resources

Each resource is represented by an instance of the resource entity, *a reusable entity that is deployed in the Grid and used to fulfil tasks submitted by Grid users*. Each entity differs from other resource entities according to the following properties:

- The number of Machines in each resource:
- The number of PEs inside each Machine.
- The speed of each CPU or processor, measured by MIPS.
- The cost of each processing unit.
- The resource allocation policy, one of the following two policies:

- Time-shared allocation policy.

Figure 28: Flow Diagram of Time-shared resources (CLOUDS Lab 2010)

- Space-shared allocation policy.

Figure 29: Flow Diagram of Space-shared resources (CLOUDS Lab 2010)

- Local load factor.
- Time Zone where the resource is located.
- Operating system
- System architecture

7.4.3.4. Grid Information Service (GIS)

Each Grid Information Service (Figure 30) is represented by an instance of a GIS entity. The Grid Information Service only provides basic operational communication with users and resources in the GridSim package and have been given a new role in BGQoS.

7.4.3.5. I/O Entities

Each I/O is represented by an instance of the I/O entity. I/O entities are responsible for the flow of information between other entities in GridSim. Since each one of these entities runs in parallel in its own thread, it is worth noting that for that reason GridSim operates at full-duplex. In addition, I/O entities have buffers, which allow the modelling and simulation of delays.

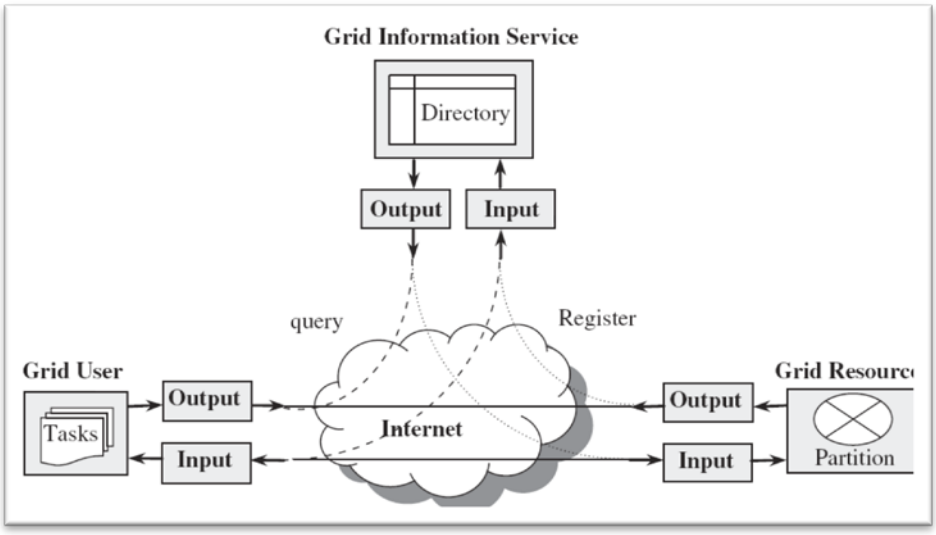


Figure 30: Grid Information Service

7.4.3.6. Gridlets

As mentioned before, user tasks in GridSim are represented by Gridlet objects. Gridlets contain the information related to the tasks, such as the size of the file sent from the user to the resource, and the size of the file that is to be returned from the resource to the user. Each Gridlet also contains information about the user that originated the Gridlet, the start time, finish time, total completion time, current status and other information.

7.4.3.7. Communication and Interaction Between Entities

All interactions between entities are carried out in the form of messages or events. These events can be initiated by an entity to be delivered either with immediate effect or with a specifically defined delay to other entities. Events are different types:

- Internal Events: Events destined to the entity itself.
- External Events: Events destined for other entities.
- Synchronous Events: The source of the Event pauses until the Event is delivered to its destination.
- Asynchronous Events: The source of the Event continues its regular operation without pausing until the Event is delivered to its destinations. All internal Events are of this type.

7.4.4. Main GridSim Classes

- GridSim:

  Responsibility: Initializing and starting the simulation. To do so the following methods are used: init ( ) and startGridSimulation ( ); both static methods. This class also activates the simulation kernel in SimJave2 and is required before any entity creation.

- GridSimCore.

  Responsibility: Management of I/O operations of an entity.

  This class is a new addition to the GridSim toolkit, aiming at taking over I/O operations: reducing the complexity of the GridSim class. Moreover, entities in this class are capable of knowing the bottleneck of a network route using the Gridsim.net package, as explained by Sulistio et al (2007).

- TrafficGenerator.

  Responsibility: Generations of network traffic.

  This is used by entities of the GridSimCore class to determine bottlenecks of routes in a network topology.

- Gridlet:

  Responsibility: The creation of *Gridlets.*

  As explained above, Gridlets are the entities in GridSim that represent user tasks. The basic Gridlet class - before modification - contains information on the tasks submitted, including, task length and number of PEs.

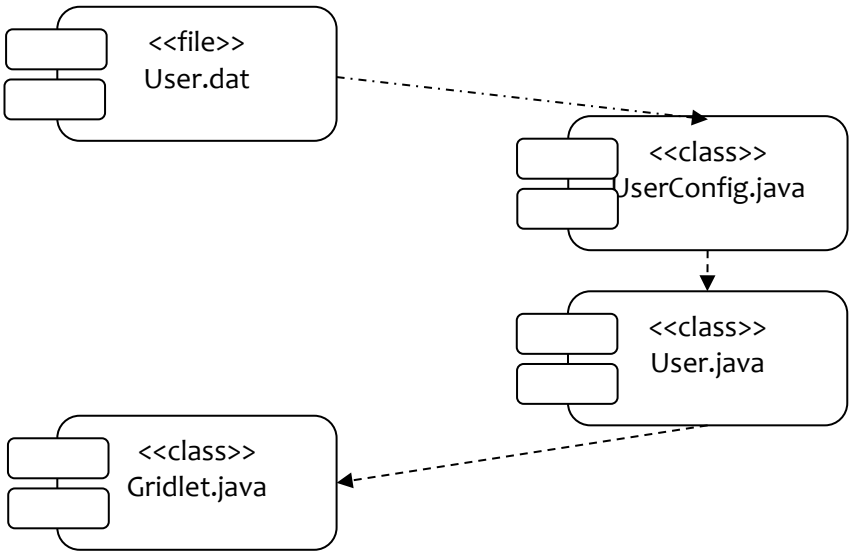


Figure 31: Component diagram for creating Gridlet in GridSim

- GridUser:

  Responsibility: The creation of *user* entities.

  This class allows the users to communicate with and register with a GIS. It allows the user to query the GIS on resources available.

- GridResource.

  Responsibility: The creation of a *resource.*

  This class represents a resource. The pre-modified version of this class includes the following properties: Time zone, scheduling policy, number of PE and their ratings. A

more recent version of this class has allowed more flexibility in the creation of different types of resources. However, this class has been modified for the purpose of this research.

- AllocPolicy:

  Responsibility: Handling the internal resource allocation policy for a GridResource.

  New scheduling algorithms can be added by extending this class.

7.4.4.1. Advanced Reservation Classes

The most recent version of the GridSim toolkit includes Advanced Reservation variations of the classes introduced above, such as: ARGridresource and ARPolicy. The addition of these classes, has allowed GridSim to expand its simulation capabilities to include:

- Requesting reservations of PEs.
- Creating reservations.
- Committing reservations.
- Modification of reservations.
- Reservation cancellation.

7.5. Modification to the Original Package

In order to evaluate BGQoS it was necessary to expand the capabilities of the GridSim to add new features to support the components implemented with BGQoS. This will help in making GridSim a more realistic simulation package that can be used with a wider range of Grid applications and can support a broad range of QoS requirements, which may vary according to differing circumstances of users, even when running the same task. Users will be able to input more specific QoS requirements. The introduction of databases into the simulation process will allow automated renegotiation to be simulated. GridSim has been developing since its initial release. GridSim 5.0 allows the users to develop their own scenarios, schedulers and allocation policies. These, in addition, to the recently added advanced resource reservation and failure detection capabilities have given the simulation package new dimensions.

The successful migration of Grid computing from the purely scientific and research domains into the business-oriented service marketplace relies on the delivery of the QoS explained in previous chapters. Therefore a clear relationship must be identified between the users of these resources and services and those who provide them. These relationships are governed by electronic contracts between them.

7.5.1. Tasks (Gridlets)

A new Gridlet class has been created with new characteristics where the user will be able to specify, create and describe their tasks in more detail. Since our model is based on QoS, the user is now able to clearly specify the QoS that each task should be allocated. Moreover, a deadline parameter has been added to the Gridlet which allows the simulation to remove the Gridlet and report it as a failure after a specific interval has passed. The Gridlet characteristics that we have added correspond to the QoS list that we have outlined in chapter 4 (4.7) and are as Shown in Figure 32.

```
public class Task extends Gridlet implements Serializable, Cloneable {

        private String originalQoSBusId;
        /** required parameteres */
        private float reliability;
        private double arrivalTime;
        private String globalTaskID;
        private double bandwidth;
        /** number of required CPUs*/
        private int PE; //
        private double memory;
        private float availability;
         /** MIPS rating of a machine used to compute estimated comp. length */
        private double estimatedCompuationalMIPS;
```

Figure 32: New List of Gridlet Characteristics

7.5.2. Users

A new user class has been added to represent the GRCs within our BGQoS with each GRC identified by their GRC_{ID}. The GRC_{ID} is also implemented into the expanded Gridlet class, allowing the model to identify the origins of each task and link it to a specific GRC, using that information to carry out its various operations.

```
public void setUserID(int userID) {
        BGQoSTask Task = this.getTask();
        Task.setUserID (userID);
        this.userID = userID;
        this.getTask().setUserID(userID);
}
```

Figure 33: Initiating the User

A user representing a GRC creates Gridlets representing tasks, which are submitted to resources in order to be executed. This operation is carried out through the Broker entity within BGQoS with its scheduling techniques outlined in the following sections, as in the example in the following Figure:

```
this.userID = Task.getUserID();
this.setTaskLocalID(Task.getGridletID());
this.setTaskStatus(Task.getTaskStatus());
this.setComputationalLength(Task.getTaskLength());
this.setTaskFinishedSoFar(Task.getTaskFinishedSoFar());
this.setCompletitionFactor(Task.getTaskFinishedSoFar()
this.setTask(Task)
this.setDeadline(Task.getDeadline());
this.setUserTier(Task.getUserTier());
this.setNumPE(Task.getNumPE());
```

Figure 34: Task information association with User

A multi-tier GRC architecture is an important element in BGQoS and is implemented within this expansion. This has been done by altering the priority methods within GridSim to accommodate the multi-tier architecture. Within this thesis, three types of GRCs have been used; therefore, three types of priorities have been implemented.

For each simulation, a GRC entity must be created and must perform a registration process before being able to submit Gridlets. Once this has been completed, each GRC may create their Gridlets, request the QoS parameters for these Gridlets and submit an execution request to the scheduling entities which carry out the operations of BGQoS as explained throughout this thesis.

Our expansion only supports deterministic fixed types of QoS, where each QoS is specified by a specific metric corresponding to the type of QoS and according to the specification we have set.

7.5.3. Resources

One of the main challenges in expanding and extending GridSim was that of expanding resources. New characteristics to model the different QoS that have been introduced and have been added, however, two main types of resources which BGQoS supports must be differentiated; computational resources and storage resources. Instrumental resources and Expertise are beyond the scope of this research.

Two additions have been made to accommodate the two different types of resources; the computing resource entity and the storage resource entity. Each entity supports the QoS that are relevant to them and have been installed in the to the simulation package to model a real life environment where the user should not worry about the underlying complexities of differentiating between the resources they require, the QoS they require and information fed back to them.

Another important addition to the original toolkit was the addition that resources can communicate: the list of Gridlets that are running; the number of Gridlets completed, and, the number of Gridlets that have failed, back to the user and/or scheduler. This dynamic retrieval of information that is sent back to the scheduler and stored is a vital process for the successful calculation of dynamic QoS which is an important and novel part of our QoS model. These additions allow us to model and simulate this process.

```
public  ResourceInfo(ResourceCharacteristics resource)      {
        this.resource = resource;
        this.numOfTotalPE = resource.getNumPE();
        this.finishTimeOnPE     = new double[resource.getNumPE()];
        this.startTimeOnPE      = new double[resource.getNumPE()];
        this.TaskListInExecution = new LinkedList();
        this.TaskListInSchedule  = new LinkedList();

 }
```

Figure 35: Resource Info

7.5.4. Scheduler and Rescheduler

The scheduler entity is responsible for the resource operations; discovery, selection and allocation. This scheduling process is QoS driven within BGQoS as expressed throughout this thesis. In order to allocate the resources successfully, a new scheduler is developed. This scheduler conforms to the model that we have presented and provides a new addition where the pre-allocation, the allocation and the post allocation process for resources are addressed. Moreover, we introduced criteria that should be measured. These criteria can be used for QoS specification. They can also be used as metrics during the processing of a task to ensure that the processing is being carried out according to agreed QoS.

In addition, a new method for dynamic renegotiation is introduced. Information on the execution of any task is retrieved over specific intervals during the allocation period. A new entity is introduced for this purpose, called a re-scheduler.

Receiving GRC execution request and $QoS_{description}$ there are events that arrive at the scheduler: initializing a request session; resource information retrieval; Gridlet scheduling, and, Gridlet dispatching.

7.5.5. Databases

The introduction of databases into the simulation process has been proposed for all phases of the simulation run, adding to the realistic execution of application runs in a simulation environment because it provides a method where a database can be populated with variable resource information that reflect the unpredictability of resource failures and resource performance in real executions. GRC information, including their IDs, tiers and virtual Organizations are kept in these databases. Moreover, GIS information and SLA templates are also held in these databases.

The characteristics for every one of the resources that are advertised by the resource providers will also be held in the databases have been referred to as RRs throughout this thesis, as well as, the success and failure rate for every specific resource over a specific number of runs. This will aid in measuring the qualitative QoS of the resource before it is allocated to another user and other tasks. The resources IDs are kept in a table inside the GIS where they register, and if they are available for global use, their IDs are registered with the global GIS. These IDs are used as pointers to search the database of resources and retrieve the information on the resource with the matching IDs maintaining up-to-date information on each resource at a specific period of time.

These databases also hold the Service Level Agreements themselves, in case they need to be referenced according to a pre-stated condition between the user and Service Provider, such as the failure to deliver a specific percentage of the QoS required. If a breach of contract occurs, then renegotiation is invoked and referencing the original agreement is important, before a new agreement is reached and replaces the original one. Figure 36 presents a sample of the database tables stored

GRC ($\underline{GRC_{ID}}$, GRC_{Name}, GRC_{Tier})
GRCTier ($\underline{GRC_{Tier}}$, Description)
Tier_Permission (GRC_{Tier}, $\underline{Permission_{ID}}$)
Permission ($\underline{Permission_{ID}}$, Description)
GRP ($\underline{GRP_{ID}}$, GRP_{Name}, $GRP_{Address}$)
Resource ($\underline{Resouce_{ID}}$, $Resource_{Type}$, $Resource_{domain}$, $Resource_{description}$, GRP_{ID})
Agreement ($\underline{Agreement_{ID}}$, $Agreement_{Name}$, $Template_{ID}$, $Application_{ID}$, GRC_{ID}, GRP_{ID})
Application ($\underline{Application_{ID}}$, GRC_{ID})
Broker ($\underline{Broker_{ID}}$, $Broker_{Address}$)

Figure 36: A portion of the database tables

7.5.6. Monitoring Tasks

This extension allows the simulation to be tracked in terms of each Gridlet individually, retrieving information on individual tasks:

```
total_count_task_generated += count_task_generated.get();
total_count_task_submitted += count_task_submitted.get();
total_count_task_scheduling += count_task_scheduling.get();
total_count_task_processing += count_task_processing.get();
total_count_task_transferred += count_task_transferred.get();
total_count_task_executed += count_task_executed.get();
total_count_task_suspended += count_task_suspended.get();
total_count_task_failed += count_task_failed.get();
total_count_task_db_stored += count_task_db_stored.get();
```

Figure 37: Task Monitoring

This includes information on the number of tasks executed successfully, the number of tasks pending, the number of tasks submitted and the number of tasks that have failed.

7.5.7. Agreement Properties

Another extension implemented is tailored to accommodating the type of agreements that are presented earlier in this thesis. Each agreement must have a specific id associated with it which identifies it. Each agreement must also include the guaranteed properties that must be met during the execution of the tasks which are stored in an array.

```
…
agreementProperties.setAgreementId("AgreementID_" + UUID.randomUUID().toString());

…
agreementProperties.addNewContext().setServiceProvider(AgreementRoleType.AGREEMENT_RESPONDER);
```

```
GuaranteeTermStateType[] guarateeStates = agreementProperties.getGuaranteeTermStateArray();
```

Figure 38: Agreement initiation and parameters

7.6. Summary

In this chapter the need for a simulation environment is highlighted. Simulation has been used to test and implement BGQoS. While there is a case of creating a brand new simulator, the availability of simulation software and toolkits and the ability to expand and extend them in order to carry out the appropriate funcationality has been chosen. GridSim was selected as the most appropriate toolkit to use and was expended with a number of new constructs in order to demonstrate the ideas embodied in BGQoS. This chapter described the expansion.

CHAPTER 8: COMPONENT EVALUATION OF BGQOS

8.1. Introduction

This chapter introduces the experimental evaluation of BGQoS. The evaluation investigates the behaviour and performance of the separate operations and components within BGQoS, and moreover, it presents an investigation and comparison between the different operations and their effect on the full model.

Each section within this chapter represents an independent experiment with a main goal. The goal, experiment metrics, experiment setup and results are explained for each section. The purpose is to illustrate the different functionalities of BGQoS that have been explained in this thesis and identify their validity, and whether they achieve their goals.

Each section contains an analysis of the experiment included within. A complete analysis and concluding remarks are presented at the end of the chapter. Chapter 9 contains an experiment utilising the full model and presents the results within.

8.2. Overhead for Resource Operations

Overhead is measured to show the efficiency and feasibility of BGQoS and corresponds to the delay experienced in the different stages of operation within BGQoS. The relationship between the GRC and the GRPs providing the resources must be conducted within the shortest time possible. The main objective of measuring overhead is to examine which resource operation incurs the most overhead and to monitor the effect of this overhead on the overall operation of BGQoS and whether the level of overhead is acceptable. In the following sections, the operations for which we have measured the overhead are presented.

8.2.1. The Measured Operations and Evaluation Metrics

Overall overhead can be broken down into specific overhead measurements, each reflecting the overhead at a specific stage within the operation of BGQoS. The following sub-sections identify the major overhead that can be calculated and identifies them as metrics for the measurement of overall overhead.

8.2.1.1. $QoS_{description}$ and Execution Request Submission

Each GRC is registered to a specific tier, therefore, the overhead incurred because of the authorisation and authentication process is negligible. This is because of the method we have employed where the tier defines the privileges associated with its registered GRCs and is checked as opposed to the GRCs themselves. The task submission overhead expected from submission is calculated as $\sum T_{submission}$. It is related to the size of the tasks submitted and the number of tasks submitted and their relative information and data requirements.

The overhead for this operation can be calculated using the following equation:

$$Overhead_a = T_{auth} + \sum T_{submission} \quad \text{(Equation 8.1)}$$

Where T_{auth} the time is required to authenticate the GRC and their login and $T_{submission}$ is the time required for submitting the tasks to be executed.

8.2.1.2. Information Retrieval from Resource Repositories (RR)

Informational retrieval is carried out in response to a GRC execution request, through querying the appropriate databases containing resource information. Since the query is sent to a database, it simplifies the resource information retrieval process by eliminating the process of querying individual resources.

Resource information within BGQoS is updated at regular intervals and therefore is assumed to be up-to-date and reflecting the current state of the resources. The query returns a list of resources that fulfil the GRC requirements stated in the $QoS_{description}$.

The overhead from this operation is calculated as follows:

$$Overhead_1 = Time_{Query} \quad \text{(Equation 8.2)}$$

$Time_{Query}$ is the time required to search the information in the database. This time varies depending on the size of the database, the number of resources and the number of accessible RRs.

8.2.1.3 Resource Selection According to $QoS_{description}$

Resource selection is the operation which concludes with the selection of resources that meet GRC requirements from the resource information retrieved process which are in turn extracted from the appropriate databases. The overhead incurred at this step is directly related to the time required to compare resources and their characteristics and whether they fulfil the requirements and constraints requested by the GRC.

$$Overhead_2 = Time_{Select} \quad \text{(Equation 8.3)}$$

Where $Time_{Select}$ is the time required to complete the selection operations introduced in earlier chapters in order to confirm whether that:

$$QoS_{requested} \leq QoS_{offered}$$

8.2.1.4. Resource Ranking

Resource ranking uses the information fed in through the previous steps in order to rank the resources according to specific criteria. The overhead is calculated as the time required for completing ranking operations of candidate resources:

$$Overhead_3 = Time_{ranking} \quad \text{(Equation 8.4)}$$

$Time_{ranking}$ is the time required to rank a list of resources within the candidate resource stack in an order that meets the GRC requirements. $Time_{ranking}$ is directly related to the number of candidate resources to be ranked.

8.2.2 Experiment Setup

- GRPs and Resources

The experiment is setup of three GRPs each providing a dedicated 10 CPU cluster for task execution. For this experiment the CPUs in terms of computational power are equal. The GRC selects the number of CPUs they require, the length of time they require them for and set a cost constraint that must not be exceeded.

- The GRC and Tasks

The graph and comments below are related to the overhead calculated for a single GRC submitting multiples of three tasks shown in table 9 i.e. 3, 6, 9,..., 30 tasks. The tasks vary in size and the number of CPUs they require.

Table 9: Task Types and Requirements

	Small	Medium	Large
# of CPUs requested	28	56	128
Execution Time Required	8 minutes	21 minutes	80 minutes

8.2.3. Results

Figure 39 Represents the overhead incurred in milliseconds from the three main resource operations; Resource Information Retrieval, Resource Selection and Resource Ranking.

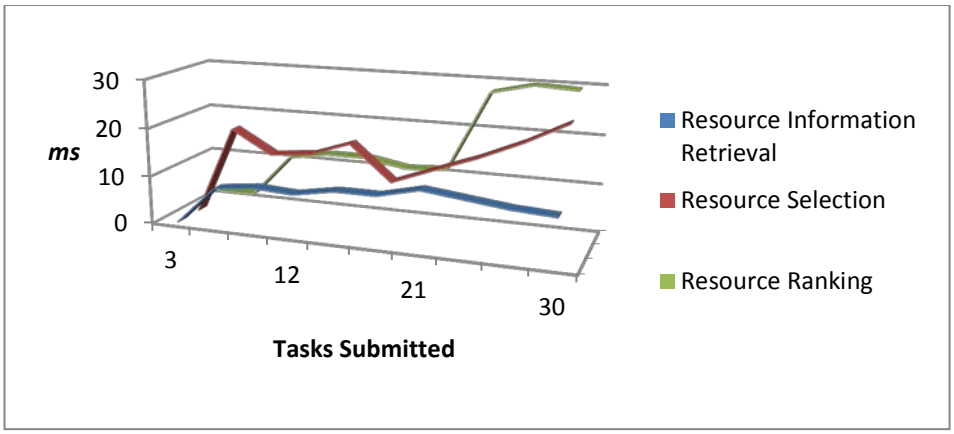


Figure 39: Resource Operations Overhead

Resource information retrieval does not depend on the number of tasks to be submitted by the GRC, with resource information retrieval accruing in a similar manner; the small variations in the Figure above are negligible. However, the overhead itself cannot be neglected, if predicted and expected. Within this experiment, it was around 10 ms.

The average overhead from resource selection was 15.5 milliseconds for the experiments carried out between 3 – 30 tasks. This was followed by a similar overhead for the resource ranking operations which averaged at 19.1 ms resulting in an average of 34.6 ms of overhead after the information has been retrieved. This overhead is to be expected and because of the main objective of BGQoS, it can be tolerated.

8.3. Overhead for Different GRC Types

In the previous section, the overhead related to resource operations has been examined under the assumption that there is a single GRC. The overhead was measured in order to establish which resource operation incurs the most overhead and whether it is tolerable and acceptable in relation to the objective of BGQoS for different numbers of tasks submitted. This section examines the overhead relative to the different types of GRCs within BGQoS.

8.3.1. Evaluation Metrics

The evaluation metric we use for this experiment is the overall overhead incurred from the resource operations, calculated for two different types of GRCs. This overhead is calculated via the following equation:

$$\text{Overall Overhead} = \text{Overhead}_a + \text{Overhead}_1 + \text{Overhead}_2 + \text{Overhead}_3 + \text{Time}_{ft} + \text{Time}_{queue} \quad \text{(Equation 8.5)}$$

Overhead_a, Overhead_1, Overhead_2 and Overhead_3 are calculated via equations 8.1, 8.2, 8.3 and 8.4 respectively. Time_{ft} is the time required for transferring execution files and Time_{queue} is the time the tasks are waiting for execution or the time that a task queues at a resource which they are allocated. FCFS queuing mechanisms are employed. Both times are calculated within the simulation.

8.3.2. Experiment Setup

- The GRC and Tasks

  Two GRCs submitting multiples of one hundred tasks shown in table 9 i.e. 100, 200, 300, 400 and 500 tasks are setup for this experiment. The tasks vary in size and the number of CPUs they require. The first GRC is of the QoS GRC (Tier A) type, the second is of the BE (Tier C) type. Moreover the Tier A GRC request includes a request for a minimum RAM of 256 associated with each CPU selected.

- GRPs and resources:

  The experiment is setup with three GRPs each providing a dedicated 10 CPU cluster for task execution. For this experiment the CPUs are assumed to have equal computational power. The QoS GRC select the number of CPUs they require, the length of time they require them for, a memory requirement, and, set a cost constraint that must not be exceeded, while the BE GRC must rely on BGQoS BE resource allocation with a predefined cost constraint.

8.3.3. Results

The results obtained for the overall overhead in resource selection for the two types of GRCs and with a 3 parameter request submitted by the QoS GRC are contained in table 10.

Table 10: QoS v BE - Overhead Difference

Number of Tasks	BE	QoS with 3 Parameters	Overhead Difference
100	21.3	185.6	164.3
200	34.0	311.3	277.3
300	46.6	385.3	336.9
400	61.3	510.4	449.1
500	74.2	769.2	695.0

The results shown in table 10 are illustrated in Figure 40.

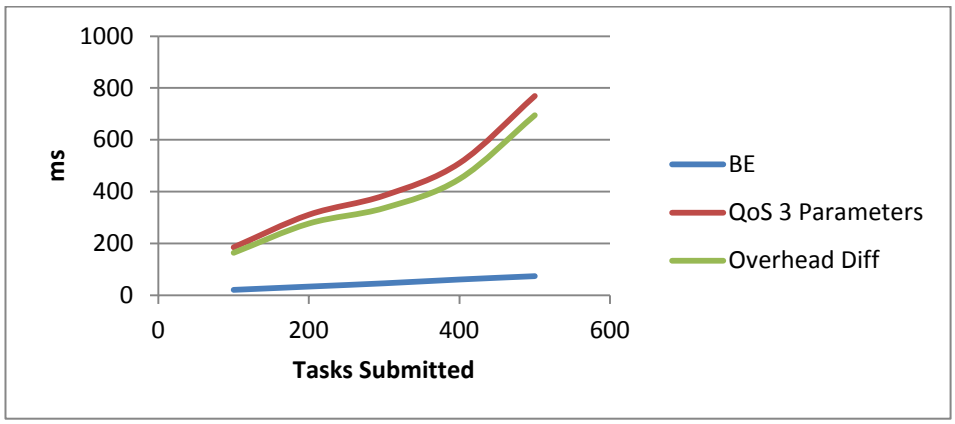


Figure 40: QoS v BE - Overhead Difference

The significant difference in the overhead incurred is due to the resource operations related to obtaining a select resource set that meet the GRC requirements. There are no resource selection and ranking operations related to BE GRCs, therefore the overhead experienced is related to resource information retrieval, $Time_{ft}$and $Time_{queue}$. While this experiment shows that these times are negligible, it is a small experiment and does not present a heavy load. No reservation was carried out and no priorities had been given.

However, this experiment has presented evidence that almost all the overhead is contained in the resource selection and ranking operations within BGQoS. Moreover, there are situations, especially where there is a significant number of resources available and low load that BE execution may reduce the makespan. BE GRCs however, cannot be guaranteed the level of QoS for which resources must be selected relative to QoS GRCs.

8.4. Locating Resources against QoS Reliability Parameter

In this section the capability of BGQoS in locating the correct resources that meet the reliability parameter requirement as requested by the GRC. Two experiments are carried out, the first without taking any constraints into consideration and only focusing on reliability. The second takes constraints into consideration and determines the effect they have on meeting GRC requirements.

8.4.1. Evaluation Metric

The reliability of a resource is a dynamic resource characteristic that is updated at regular intervals according to up-to-date information retrieved on the current status of the resource. Reliability *Re* as a percentage is calculated as:

$$\mathrm{Re} = \frac{\mathrm{k}}{\mathrm{n}} \times 100 \qquad \text{(Equation 8.6)}$$

Where n is the total number of tasks submitted within a pre defined period of time and k is the number of tasks that have been executed successfully, meeting the GRC's QoS requirements throughout. n = 10 for this experiment.

8.4.2. Reliability without Constraint

In this experiment the reliability request is made as a sole QoS parameter, with no cost constraint, C, or time constraint, T, playing a role in resource selection.

8.4.2.1. Experiment Setup

A database has been populated with a set of resources with variable reliability information. The resources had been used to carry out 500 mock tasks in which the failure rate has been random. This allowed the resource information to be updated and therefore, the information associated with the resources stored within the database represents a simulated real time information model. A single GRC submits 100 tasks with reliability requirements of 70%, 80%, 90% and 95 %.

The experiments have been carried out using a populated database of between ten and seventy, using increments of ten after each run.

8.4.2.2. Results

Figure 41 illustrates the number of resource that met the GRC reliability requirements, for each experiment:

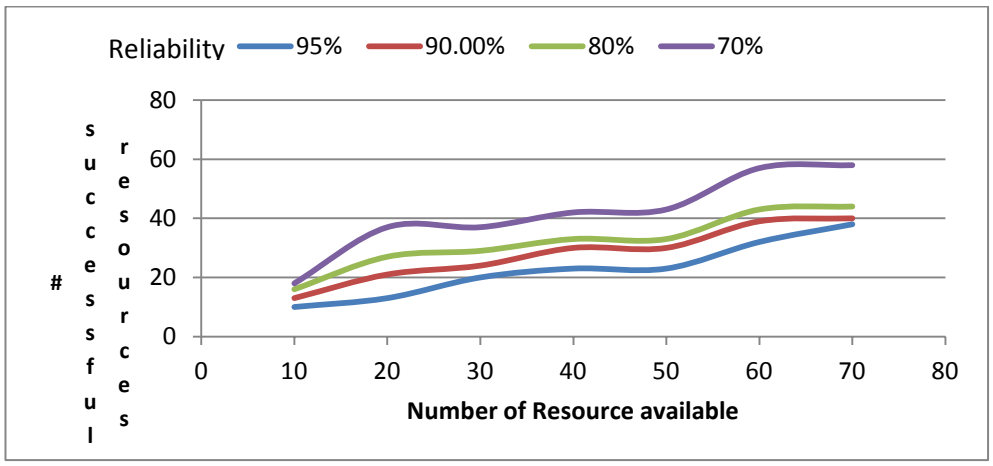


Figure 41: Successful requests - Reliability

The number of resources meeting the request by the GRC increased when the number of resources that are available within the RR increased and increased when the reliability requirement requested by the GRC decreased. Overall, for a single parameter, BGQoS was capable of locating the appropriate resources meeting the GRC requirement of reliability. The next subsection takes into consideration the cost and time constraints and their effect on the number of resource selected.

8.4.3. Reliability with Constraints

This experiment measures the effects of C and T on the reliability request as a QoS parameter in the request.

8.4.3.1 Experiment Setup

A database has been populated with a set of resources with variable reliability information. The resources had been used to carry out 500 mock tasks in which the failure rate has been random. This allowed the resource information to be updated and therefore, the information associated with the resource stored within the database represents a simulated real time information model. A single GRC submits 100 tasks with reliability requirements of 80 % and different sizes illustrated in table 9.

The experiments have been carried out using a populated database of 10 to 170 resources incremented by 10 at each run. Each resource is associated with a randomly generated price ranging from 500 to 2500 units per unit of time, and randomly generated time constraints T ranging from 500 to 1500 time units and all the resources are of equal computational power.

8.4.3.2. Results

Figure 42 illustrates the results obtained, representing the total number of resources that have been returned as meeting the GRC request and within the time and cost constraints set.

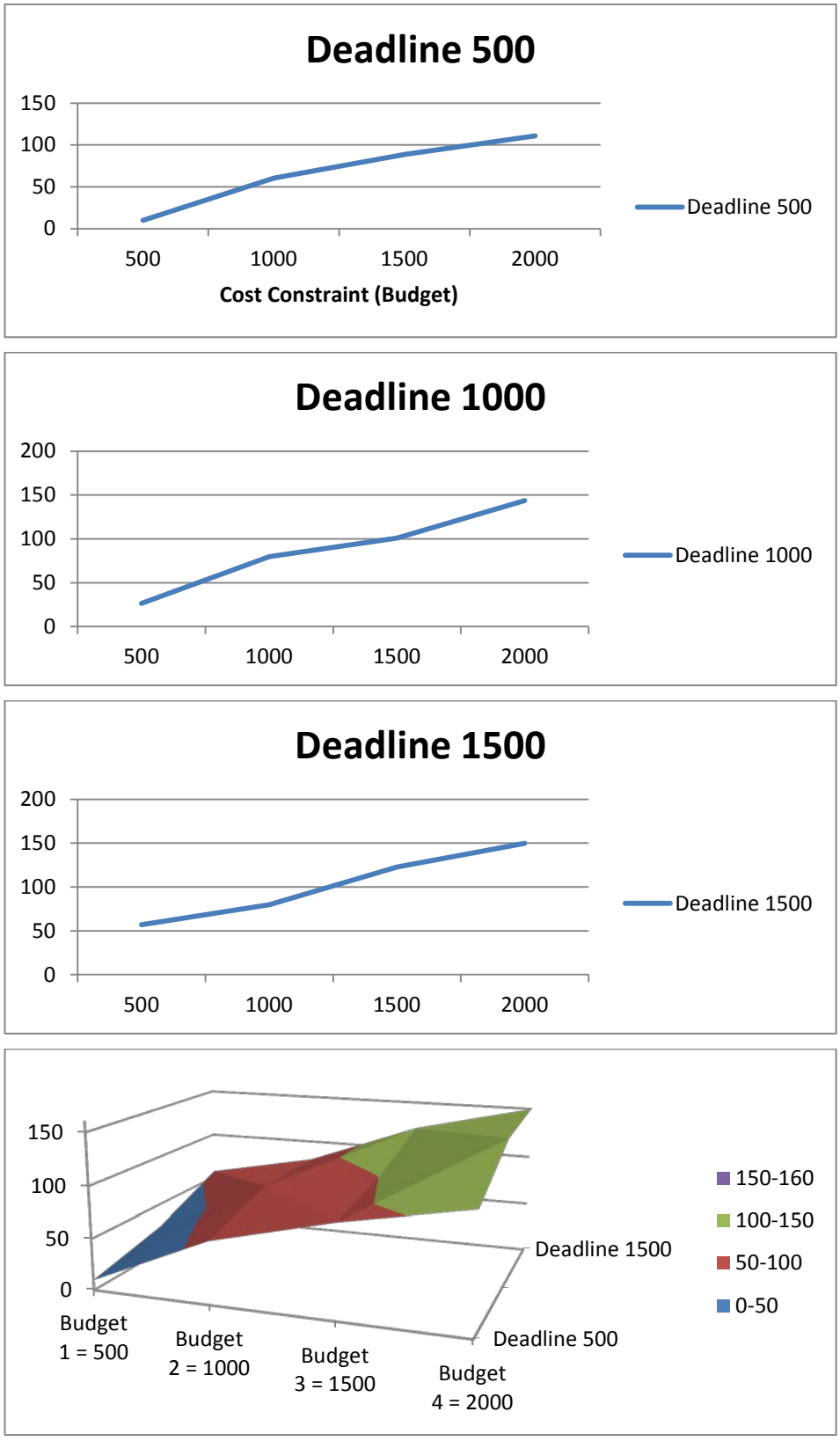


Figure 42: Effect of Budget and Deadlines - Returned Resources

The experiment intends to simulate a real world environment such that the more expensive the resource, the higher the level of QoS expected in return. Random generation of resource cost, as well as time and cost constraint aim to simulate different scenarios where these two factors make a difference in relation to the number of resources that can be considered as candidates.

The results obtained show that there was an increase in the number of potential resources returned with an increase in C. On the other hand, the number of candidate resources has increased with an increasing T.

8.5. Resource Selection

Resource operations have been explained throughout this thesis. Earlier in this chapter the overhead related to these operations has been evaluated and presented. This section builds on the evaluation carried out in the previous sections of this chapter. The resource selection process concludes with each resource bring assigned a rank. The highest ranked resource set is to be selected, however, in real scenarios; this may not be the case. Many factors such as unexpected failures, resource degradation, dynamic availability information and policy mismanagement can result in that the top ranked list is not selected. This section presents the simulation carried out for evaluating this process and identifying the rank of the resources selected by the BGQoS to execute the tasks.

8.5.1. Experiment setup

- GRPs and resources:

The experiment is set up with 9 GRPs each providing a dedicated 10 CPU cluster for task execution. For this experiment the CPUs in terms of computational power are equal. A QoS GRC may select the number of CPUs they require, the length of time they require them for, a memory requirement and set a cost constraint that must not be exceeded, while a BE GRC must rely on BGQoS BE resource allocation with a predefined cost constraint.

- The GRC and tasks

The results and comments below are related to single QoS GRC submitting 100-1300 tasks submitting multiples of 200 tasks shown in table 9 i.e. 100, 300 ,500 ,700,…,1300 tasks. The tasks vary in size and the number of CPUs they require.

8.5.2. Results

Figure 43 shows the results that have been obtained from the experiment above. While most successful requests have been met with resources of rank 1, there are a significant number of resources ranked 2nd, 3rd and 4th selected, due to random resource failures implemented within the experiment. More importantly, BGQoS has managed to maintain successful QoS driven selection throughout, by selecting the highest ranking, most appropriate and available resource set.

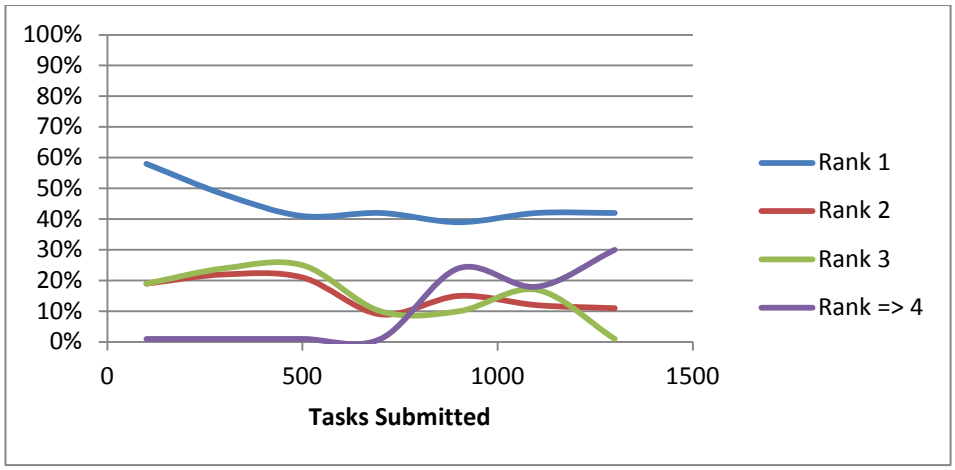


Figure 43: Rank Selected Percentage

It also shows that the BGQoS ranking criteria provide an alternative if the highest ranked set of resources cannot be selected or is not the most preferred according to the policy matchmaking process between the GRC and GRP, for example. BGQoS has performed that successfully, while delivering the requested level of QoS.

8.6. Effect of GRC Type on Successfully Completed Tasks

This section shows the results of the simulations with the aim to evaluate the success of a GRC request according to the GRC type and the tier they belong to and whether they have access to global resources.

8.6.1. Evaluation Parameters

The percentage of successfully executed tasks is the metric for this experiment and can be calculated by equation 8.7:

$$\frac{\text{Successful tasks}}{\text{Total Tasks}} \times 100\ \% \qquad \text{(Equation 8.7)}$$

8.6.2. Experiment Setup

- GRCs:

Two types of GRCs are represented, QoS GRCs and BE GRCs split into a 40-60 percentage ratio. This is done in order to give BE GRCs a numerical advantage and explore the effects of that on the task completion ratio of the more privileged GRCs.

The QoS GRC within this experiment reflects the tier A GRC. The BE GRC represents a tier C GRC. The QoS GRC may select the number of CPUs they require, the length of time they require them for, a memory requirement and set a cost constraint that must not be exceeded, while the BE GRC must rely on BGQoS BE resource allocation with a predefined cost constraint.

Each GRC submits 500 tasks in their execution request.

- Resources:

The experiments have been carried out using a populated database of 10 to 110 resources incremented by 10 at each run.

8.6.3. Results

Figure 44 illustrates the results obtained:

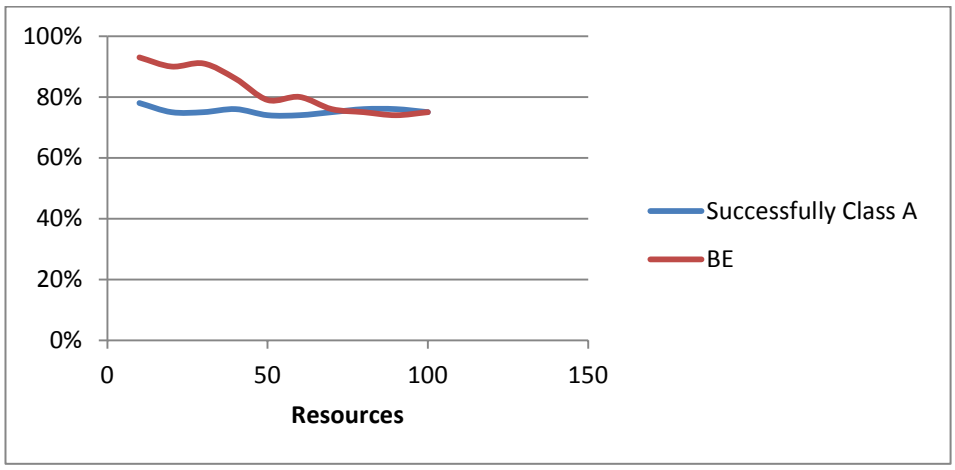


Figure 44: Successful Task Percentage - Class A v BE

From Figure 44, it is noticable that while BE effort services have performed well when the number of tasks was low, the tier A GRC was capable of achieving a higher percentage of successful task execution according to their requirements. This is due to better resource management, resource reservation services and reallocation services available to the GRC of this tier. This can be expanded to very large task execution requests, where the percentage of tasks executed will remain high and verifies the necessity for a model that guarantees a stable rate of task execution throughout.

8.7. GRC Access Authorisation

This section presents the results of an experiment carried out in order to examine the differences in the success rate of task completion with relation to which resources the GRC has access to; local, partner or global.

8.7.1. Evaluation Metric

For this experiment, the percentage of successfully executed tasks can be calculated by Equation 8.7.

8.7.2. Experiment Setup

- GRCs:

  Two types of GRCs are represented, QoS GRCs and BE GRCs split into a 40-60 percentage ratio. This is done in order to give BE GRCs a numerical advantage and explore the effects of that on the task completion ratio of the more privileged GRCs.

  The QoS GRC within this experiment reflects the tier A GRC. The BE GRC represents a tier C GRC. The QoS GRC may select the number of CPUs they require, the length of time they require them for, a memory requirement and set a cost constraint that must not be exceeded, while the BE GRC must rely on BGQoS BE resource allocation with a predefined cost constraint.

  Twenty GRCs (12 QoS GRCs and 8 BE GRCs). Each submits 10 to 1300 tasks incremented by 10 at each run. Only QoS GRCs have access to non-local resources.

- Resources:

  The experiments have been carried out using a populated database of 120 resource, 120 partner resources and 120 global resources. Each of these resources is assigned random computational power equivalent to between 2.0 GHz and 3.2 GHz and Memory between 256 KB and 2GB of RAM.

8.7.3. Results

Figure 45 illustrates the results obtained:

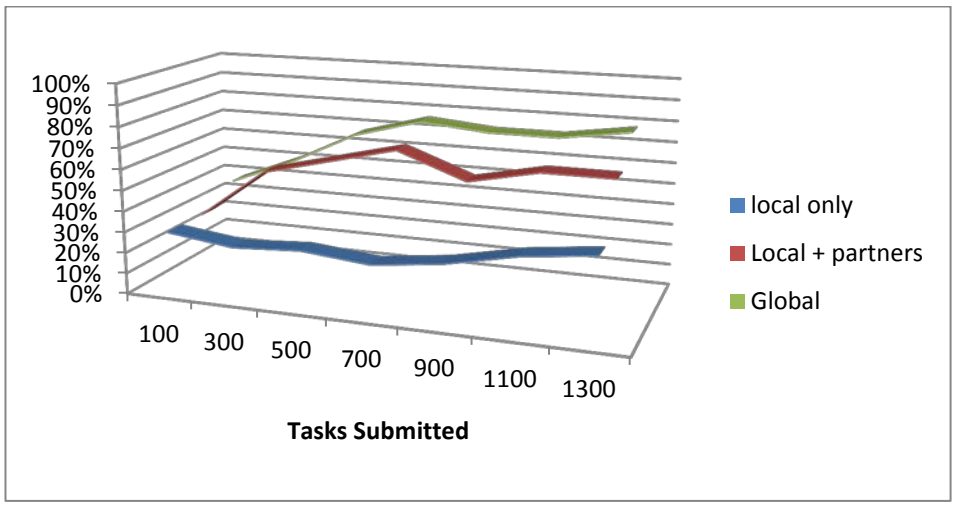


Figure 45: Successful Task Percentage - Local Access, Partner Access, Global Access

From Figure 45 it can be deduced that while there is a significant difference between case 1 (local access only) and case 2 (local + partner) and case 3 (local + partner +global), the difference between case 2 and case 3 is not as significant, even though access to global resources represents a significant rise in the number of resources available. These results further emphasise the importance of a multi-tier GRC architecture where resource access is managed and facilitated while maintaining a balance between access and effectiveness.

8.8. Processing Time for Different GRC Types

This section examines the differences between processing times for different types of GRCs. Table 11 contains the results of running 10 Simulations, running 25, 50 and 75 requests of 150 tasks each with QoS request of at least 50 MIPS assigned per request.

Table 11: BE v QoS

# of requests	QoS processing time (Tier A)	Total MIPS available	BE Processing time (Tier C)
25	22.4	10000	117.8
50	46.56	10000	118.45
75	125.56	10000	301.64

The table clearly illustrates the difference in processing time between the two types of GRCs and the substantial processing time gained by the QoS GRCs as the number of tasks submitted increases. Moreover, while BE GRCs with no time constraints or time constraints have completed their tasks, they have not interfered with the completion of QoS GRC tasks. This successfully illustrates that BGQoS maintains the advantage for QoS GRCs while maintaining that BE tasks are executed successfully.

For further comparison, the same experiment has been carried out against the traditional First Come First Served (FCFS) approach. Table 12 presents the results of running 10 Simulation.

Table 12: BGQoS v FCFS with QoS GRCs

# of requests	BGQoS processing time	Total MIPS available	FCFS Processing time
25	27.9	10000	122.28
50	56.118	10000	136.85
75	156.98	10000	240.1

BGQoS outperformed FCFS for all task numbers while providing QoS guarantees successfully. However, FCFS did out perform our BE GRCs within BGQoS for the same number of tasks and resources.

8.9. Effect of the Number of QoS Parameters Requested

The number of parameters submitted by the GRC with the execution request in the $QoS_{description}$ vary. These parameters are used to locate the appropriate resources. This section examines the effect of the number of parameters submitted on the resource operations and success rate.

8.9.1. Experiment Setup

A database has been populated with a set of resources with variable reliability information. The resources had been used to carry out 500 mock tasks in which the failure rate has been random. This allowed the resource information to be updated and therefore, the information associated with the resource stored within the database represents a simulated real time information model. A single GRC submits 100 tasks with different sizes, explained in Table 9:

The experiments have been carried out using a populated database of 10 to 120 resource incremented by 10 at each run. Each of these resources is assigned random computational power equivalent to one of the following values 2.0, 2.4 and 3.2 GHz and Memory between 256 MB, 512 MB and 2GB of RAM.

- GRC requests:

The experiments ran with 1, 2, 3 and 4 QoS parameters requested by the GRC. These required parameters were as follows:

- 1 Requirement → CPU 2.4 GHz.
- 2 Requirements →CPU 2.4 GHz, RAM 512 MB
- 3 Requirements →CPU 2.4 GHz, RAM 512 MB, Reliability 80%
- 4 Requirements →CPU 2.4 GHz, RAM 512 MB, Reliability 80 %, Cost 350 units.

8.9.2. Results

The following two figures, Figure 46 and Figure 47, show the results from the two runs of the experiments that we have performed and the successfully completed tasks in each run:

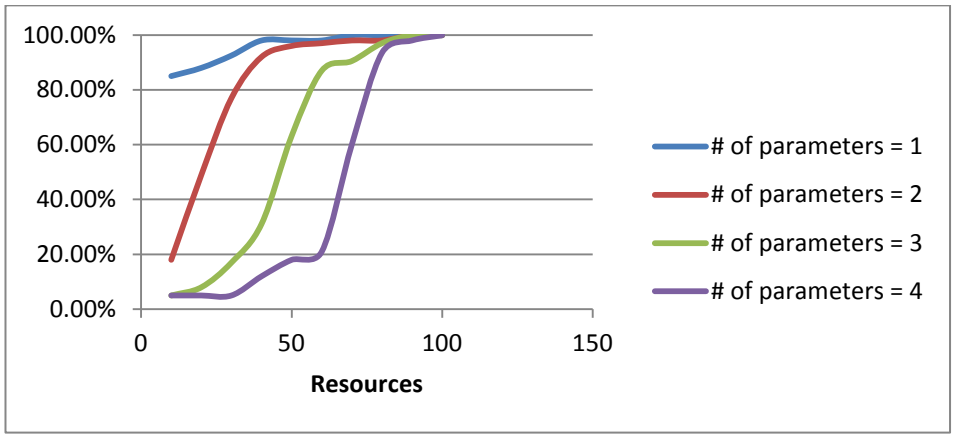


Figure 46: Successful Tasks - # of Parameters Requested

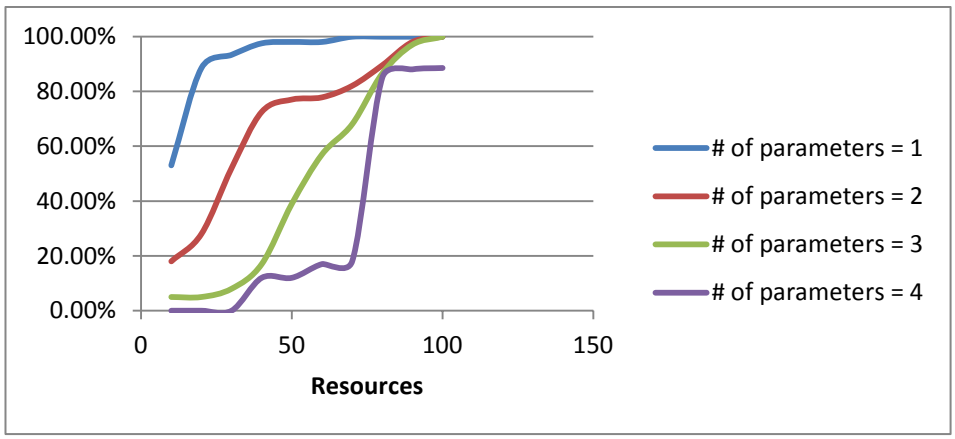


Figure 47: Successful Tasks - # of Parameters Requested (2)

The larger the number of resource available, the higher the percentage of successful requests that have been achieved, overall the percentage of successful requests is illustrated in Figure 48:

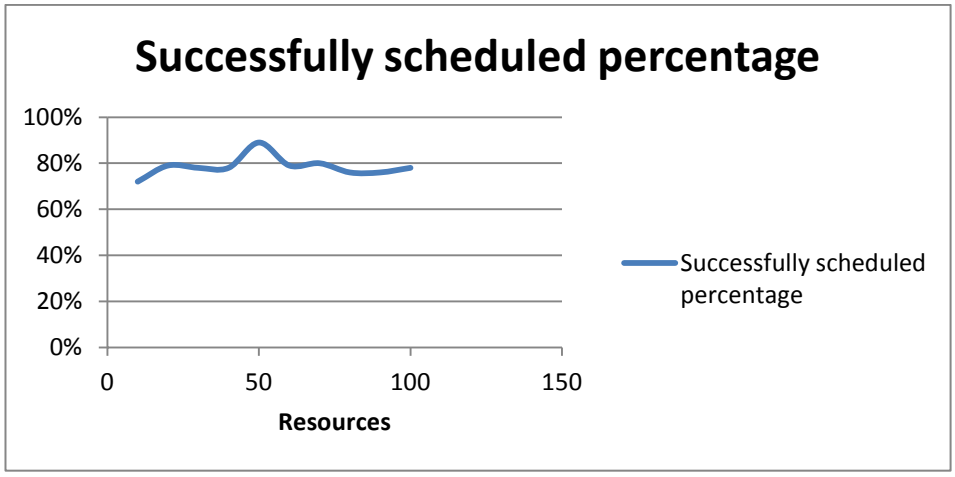


Figure 48: Successfully Scheduled Percentage

In comparison, Figure 49 illustrates the results for running the same experiment using Gridway:

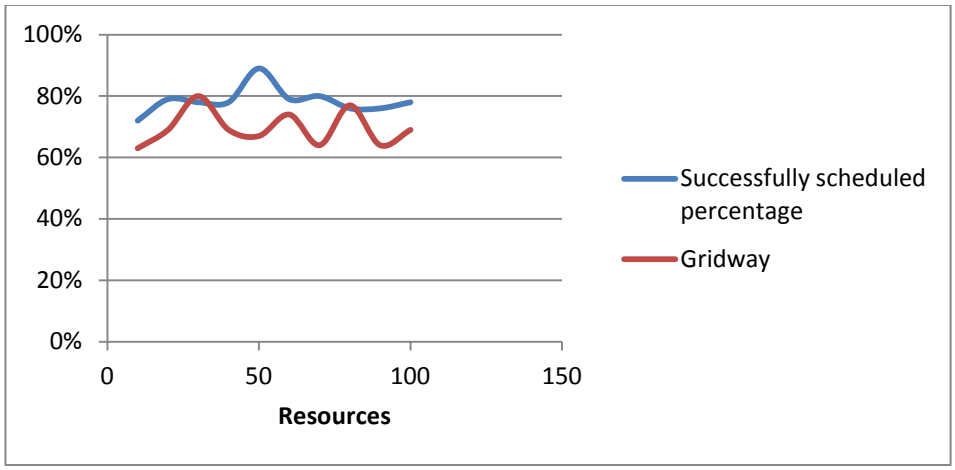


Figure 49 Successfully Scheduled Percentage - v Gridway

Overall, there has been an increase of .08 % of successfully scheduled tasks using BGQoS; this is due to a faster matchmaking process and the implementation of resource operations that tailor to the GRCs request.

8.10. Scheduling Precision

The scheduling precision is measured as the proximity of $QoS_{requested}$ in relation to the $QoS_{offered}$. The precision is an important parameter that measures the accuracy of BGQoS in selecting the appropriate resources. This section examines the precision and presents the results associated.

8.10.1. Evaluation Metric

The scheduling precision is calculated by the equation 8.8:

$$\frac{\text{QoS}_{\text{offered}}}{\text{QoS}_{\text{requested}}} \times 100\ \% \qquad \text{(Equation 8.8)}$$

8.10.2. Experiment Setup

A database has been populated with a set of resources with variable reliability information. The resources had been used to carry out 500 mock tasks in which the failure rate has been random. This allowed the resource information to be updated and therefore, the information associated with the resource stored within the database represents a simulated real time information model. A single GRC submits 100 tasks with different sizes, explained in Table 9.

The experiments have been carried out using a populated database of 10 to 120 resource incremented by 10 at each run. Each of these resources is assigned random computational power equivalent to one of the following values 2.0, 2.4 and 3.2 GHz and Memory between 256 MB, 512 MB and 2GB of RAM.

8.10.3. Results

Figure 50 presents the results of the experiment above.

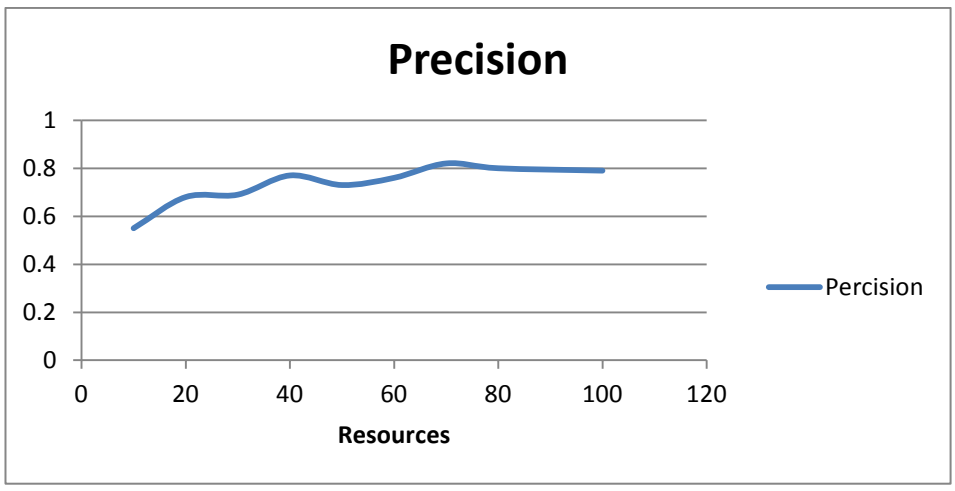


Figure 50: Average percentage of $\text{QoS}_{\text{offered}}$ in relation to the $\text{QoS}_{\text{requested}}$

8.11. Partial Offers

Section 8.10 examined the relation between the of $QoS_{requested}$ in relation to the $QoS_{offered}$. It shows that in general that there are cases where there have been a number of resource sets that partially met the GRC requirements. They are not selected within BGQoS and combining more than one solution has not been implemented, however, it is part of the future work and will be implemented then. This section presents the number of partial offers that could be retrieved and their proximity to the requested level.

8.11.1. Results

Figure 51 presents the results from this experiment.

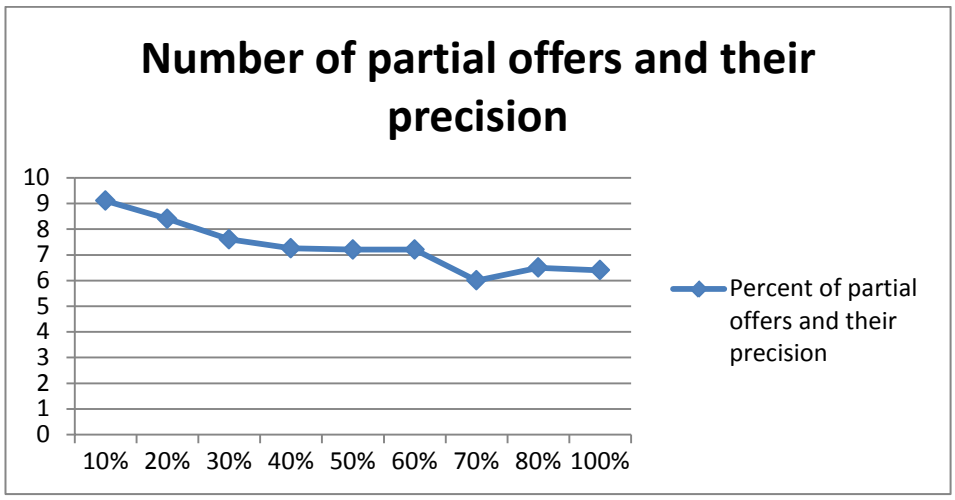


Figure 51: umber of Partial Offers and their Precision

There were a number of resources capable of providing just under the requirements set by the GRC. This may be acceptable by some GRCs and these resources can be added to potential resources for consideration in future work. Moreover, resources could be combined in order to achieve the requirements set by the GRC and this combination can lead to successful task executions.

8.12. QoS Requirements vs Resource Utilisation

The following experiment examines the relationship between meeting the QoS requirements and resources utilisation. The focus on resource utilisation is an important factor in determining whether BGQoS is a viable solution that manages the utilisation level of resource at an efficient level.

8.12.1. Evaluation Metric

Resource utilisation is a percentage within a specific period of time is calculated as:

$$U_i = \frac{O_i^t}{EO_i^t} \times 100$$

U_i is the Utilisation of Resource R_i over a period of time t. O_i^t is the actual output from R_i and EO_i^t is the estimated output from R_i over the same period of time t. None of the resources were utilised to their full capacity in this experiment.

8.12.2. Results (1)

Table 13 includes the types of requirements requested by the GRC, the percentage of QoS Delivery and the resource utilisation relative to each set of requirements using the same set of resources as explained in section 8.11.

Table 13: Successful QoS Deliver Percentage and Resource Utilisation

QoS Parameters	Successful QoS delivery	Resource Utilisation
T Constraint + # CPUs	96.8%	93.9%
T Constraint + Time requested for CPU utilisation	95.4%	93.7%
T Constraint + C Constraint	92.7%	89.3%
C Constraint + #CPUs	91.7%	93.7%
C Constraint + Time requested for CPU utilisation	89.7%	77.5%

8.12.3. Experiment Setup

The experiment is setup using the workflows of two Grids; auverGrid (Jacq et al 2008) and Grid 5000 (Grid'5000 2010). The information on both Grids is publicly available at (The Grid Workloads Archive 2007).

The aim for BGQoS is to operate within an environment where resources fail for different reasons as we have introduced before. Our experiments were all based on a set of resources R_1 .... R_i, representing different computing resources with random failures. Figure 52 represents the percentage (%) of failures for the first 5 resources over a 30 day simulated period.

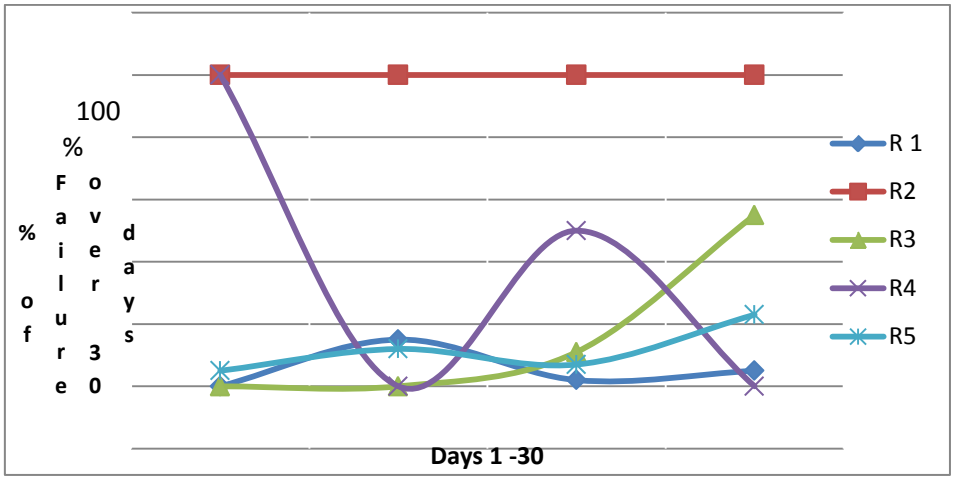


Figure 52: Failure over 30 days

From the Graph above it can be noticed that resource 2 was down for the duration of the simulation, while other resources provide their service without or with little failure over the 30 day simulated period. The information gained from this sample is stored in the RR and updated dynamically within BGQoS to represents up-to-date, relevant and accurate data on the state of the resources at any point. Decision making is improved, accurate resources sets are compiled and GRC requirements and tailored to more efficiently.

Figure 53, shows 1600 applications run over 9 resources with varying cost constraints. The Time Constraint has remained constant throughout and is set to 20 days or 480 hours. As C grows bigger the number of applications carried out rises until it reaches approximately 1000 applications with the loosest constraint.

The resources are made up of computational resources with processing capacities ranging from a minimum of 2.0 GHz to a maximum of 7.4 GHz and with a memory ranging from a minimum of 512 MB to a maximum of 8 GB.

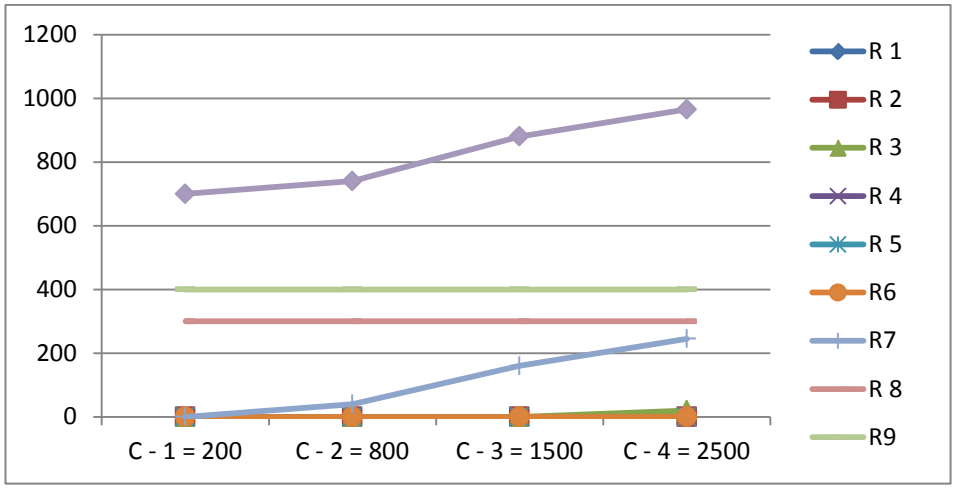


Figure 53: Resource Utilisation over 30 days

In order to portray a real environment, extra simulated execution requests have been added in order to generate competition for resource allocation and observe resource utilisation for all the resources accissble. Some resources executed tasks from the main execution request, Resource 8 and 9 ran the bulk of the tasks and are the most efficiently allocated as Figure 54 illustrate.

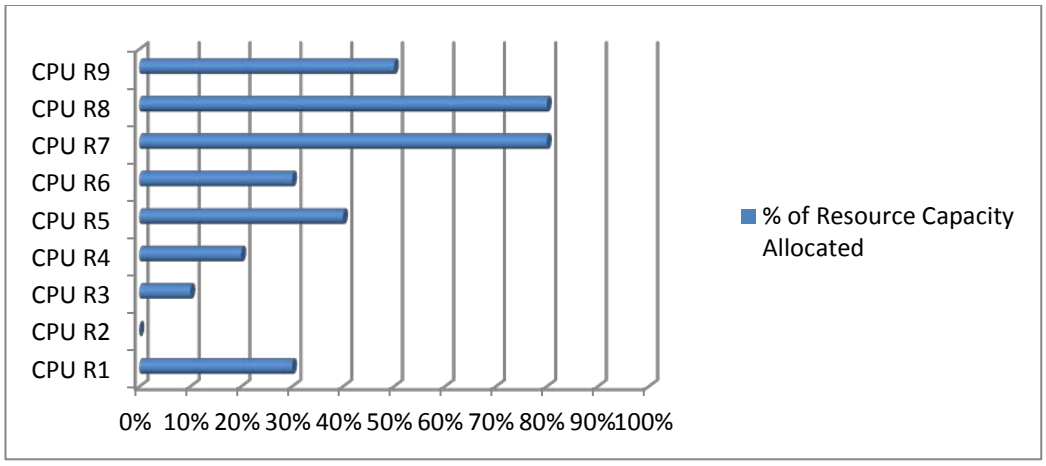


Figure 54: Allocated Resource Capacity

8.13. On Demand vs Advanced Reservation

This section examines the difference between requesting resources on-demand and reserving resources in advance.

8.13.1. Evaluation Metrics

i) Execution time which is measured by the parameter:

$$\text{Time}_{\text{execution}}$$

ii) Queuing time which measures the time spend in queues at resources, noting that BGQoS employs an FCFS queuing setup:

$$\text{Time}_{\text{queue}}$$

8.13.2. Experimental Setup

Variable resource populations between 10 and 170 resources have been used to populate a database. A single QoS GRC (Tier A) submits 500 tasks with varying parameters as in the previous sections. The experiment has been run 5 times for each number of parameters.

8.13.3. Results

Table 14, presents the results obtained from the experiments above.

Table 14: OD v AR

Exp#	# Parameters	AR $Time_{queue}$	AR $Time_{execution}$	OD $Time_{queue}$	OD $Time_{execution}$
1	1	0	681	28	605
1	2	0	852	377	748
1	3	0	942.3	245	852
2	1	0	685.2	34	360
2	2	0	854	497	335.4
2	3	0	953.2	247	704
3	1	0	685	141	412
3	2	0	833.2	473	405.6
3	3	0	943.5	537	530.8
4	1	0	698	171	414
4	2	0	863	377	419
4	3	0	946.5	292	563
5	1	0	673.6	188	369
5	2	0	852	245	486
5	3	0	943.5	488	701

Figure 55 illustrates the Table 14

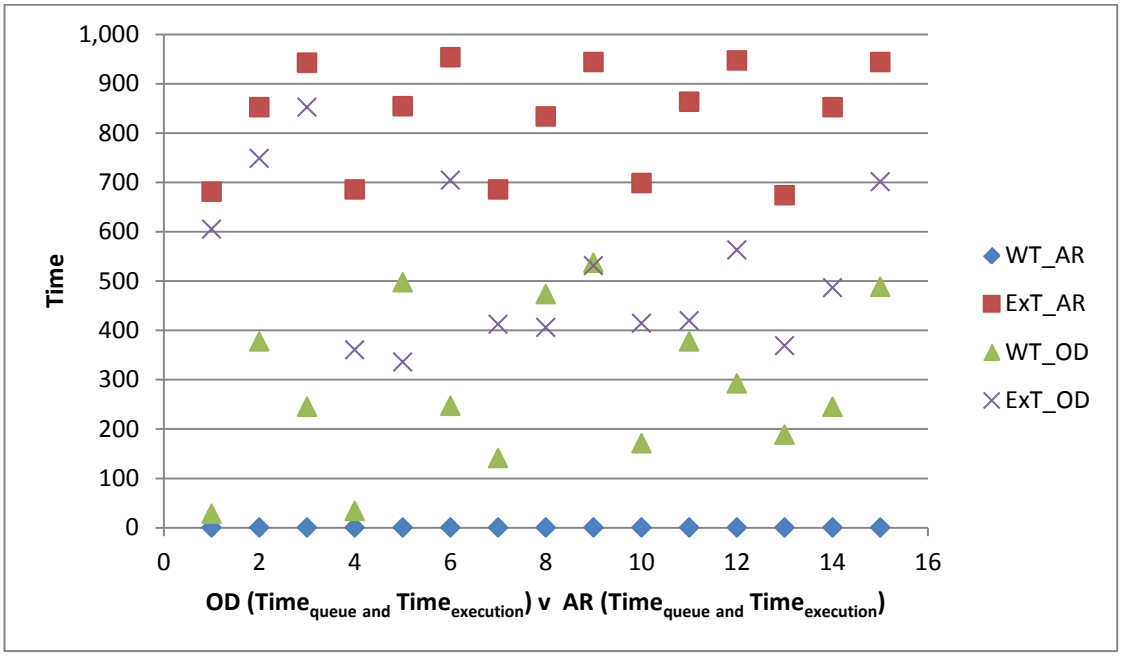


Figure 55: OD v AR

There is a small variation in$Time_{execution}$, where there are resources that become available, those are better suited and were not reserved initially delivering better performance and lowering $Time_{execution}$ in some cases. However, the elimination of waiting time for tasks with resources reserved might be beneficial to some GRCs. In some cases, however, even with the elimination of $Time_{queue}$ on-demand resource allocation has delivered better times overall, meaning that OD resource allocation is viable and feasible. Having said that, these results do depend on multiple factors including: the size of the resource population, the number of GRCs and other environmental factors.

8.14. Reallocation and Migration

Rescheduling and migration within BGQoS is carried out using a set of criteria that depends on a tolerance ratio set by the GRC, which in the case of this experiment is at 75 %. This section examines the effect of success and completion rations with and without reallocation. It also shows the effect of increasing the number of submitted tasks per minute on the level of QoS delivered with reallocation.

8.14.1. Evaluation Metrics

Metric 1:

Average increase in successful completion according to QoS level with reallocation with increasing number of tasks submitted per minute, measured by total number of Tasks Completed and calculated by Equation 8.9.

$$\frac{T_{reallocation} - T_{noreallocation}}{T_{Total}} \times 100\% \qquad \text{(Equation 8.9)}$$

Within this experiment, 20 simulation runs were carried out, each producing a random generated number of tasks as the initial number, increasing the number of tasks submitted by a specific percentage every simulated hour.

Metric 2:

Average QoS reduction with increasing the number of tasks submitted per minutes

8.14.2. Results

Table 15 shows the effect of increasing the task submission rate on the percentage of tasks completed.

Table 15: Effect of increasing Task Submission Rate on Tasks Completed

#of tasks per minute	percentage of increase in tasks completed with reallocation
+5%	0.0
+10%	24%
+20%	12%
+30%	46%
+40%	37%
+50%	35%
+60%	11%

In all of the above cases, the number of tasks completed according to the GRC requirements and within the constraints has increased with reallocation employed as opposed to considering the execution as a failure when no reallocation is available. Table 16 shows the effect of increasing the task submission rate on QoS level.

Table 16: Effect of increasing Task Submission Rate on QoS Level

#of tasks per minute	Percentage of QoS Reduction
+5%	0.0
+10%	-23%
+20%	-11%
+30%	-36%
+40%	1%
+50%	-18%
+60%	-4%

The QoS degradation associated with the increase in the number of tasks submitted per unit time is expected because of the competition that results from a larger number of tasks competing for the same number of resources in the available population, as well as the availability of resources providing a lower level QoS which are selected in order to meet the constraints submitted by the GRC. However, this degradation is tolerable and further enhanced with resource reallocation in case of failures. At one point an increase has been achieved, in this instance, the reallocated tasks were executed on resources providing a higher level of QoS than the original resources on which they would have been expected to be allocated. However, the normal situation would be for the resources providing a high level of QoS to be reserved or not available for running tasks, therefore it would be more difficult to achieve the same level or a higher level of QoS with a larger task submission rate.

8.15. Violations

This section examines the number of violations and the number of successful executions within an experiment. The purpose is to show how many violations occur in systems, where a violation is a level of QoS that does not meet the tolerance ration set by the GRC and how BGQoS handles these violations.

8.15.1. Evaluation Metrics

The following evaluation metrics were used:

- The number of requests granted without violations
- The number of violated executions
- The number of violations outside tolerance ration
- The number of requests granted without violations
- The number of violated executions
- The number of violations outside tolerance ration

8.15.2. Experiment Setup

- GRC and Task requests

  A single QoS GRC(Tier A) submits 500 tasks for execution with 1, 2, 3 and 4 parameters as explained in previous section 8.9. The GRC is associated with a QoS delivery tolerance ratio of 85%.

- GRPs

  The experiment is setup of three GRPs each providing a dedicated 10 CPU cluster for task execution. For this experiment the CPUs in terms of computational power are equal.

- Resources

  A database has been populated with a set of resources with variable reliability information. The resources had been used to carry out 80 mock tasks in which the failure rate has been random. This allowed the resource information to be updated and therefore, the information associated with the resource stored within the database represents a simulated real time information model.

8.15.3. Results

The results obtained from the experiment above are presented in table 17, where the number of executions without violations, the number of violations and the violations that required action are shown, as well as the the number of granted GRC requests.

Table 17: number of violations within and outside ratio in relation to granted requests

# of Parameters	# of executions without violations	# of violated executions	# of violations outside tolerance ration	# of requests granted
1	14	62	9	76
2	16	57	12	73
3	10	58	16	68
4	3	52	25	55

The number of violations according to the GRC ratio was surprisingly large; however, BGQoS has managed these violations well and performed a set of successful reallocation operations according to ratios, which the next section examines in more detail.

8.16. Reallocation with Ratio

This section examines the variation of ratios and parameters with reallocation operations.

8.16.1. Evaluation Metric

The evaluation metric for this experiment is whether the tolerance ratio has been met by the resources throughout the execution of tasks.

Migration decision: $\begin{cases} \text{Migrate, if DR} < TR \text{ and } ACR < AR \\ \text{No Migration , if RDR} \geq \text{TR or ACR} \geq \text{CR} \end{cases}$

The value of which is compared to the ratio associated with each experiment which is variable and specific to the experiment itself.

8.16.2. Experiment Setup

- GRC:

  A single QoS GRC (Tier A) submits 500 tasks with the following requests for the complete execution:

- Average CPU Power
- Average Memory (RAM)
- Allocated Storage

- GRPs and Resources:

  12 Clusters are simulated within this experiment. 10 are CPU clusters containing Computational resources, while 2 are Storage Clusters containing storage resources. The resource characteristics are in table 18.

Table 18: Resource types

Type	Minimum	Maximum
CPU	2.4 GHz	3.8 GHz
RAM	128 MB	2048 MB
Storage	1 GB	1024 GB

8.16.3. Results

Table 19 represents the results obtained from running 15 experiments with varying requirements on a set of resources, explained above. The table also illustrates the number of experiments in which violations beyond the ratio have occurred:

Table 19: Meeting the Ratio QoS Demand

Exp	Ratio %	Average CPU Requested	Actual Average CPU	Storage Requested	Actual Storage	Average Memory Requested	Actual Memory
1	97	7.4	7.1	20	19	256	256
2	94	8.3	7.9	20	19	256	256
3	95	24.2	24.5	20	19	256	256
4	98	33.7	32.5	150	180.3	1024	1024
5	95	31.3	28.2	150	178.8	1024	1024
6	95	32.5	29.9	280	148.5	1024	1280
7	93	186	182.3	200	231.2	1024	1280
8	86	71	80.2	25	23	256	128
9	70	112	115.9	25	26	256	256
10	60	2.2	2.4	25	26.3	256	256
11	55	8.3	8.9	25	25	256	256
12	95	5.3	7.2	25	25.8	256	1024
13	90	6.4	2.6	25	28.9	256	256
14	80	2.5	8.9	25	29	256	256
15	85	8.5	6.7	25	25	256	256

8.17. Further Comparison with FCFS

In this section, further comparison with FCFS is provided. The value of this comparison is that it shows not only that BGQoS delivers a level of QoS that is requested by the GRC but also executes tasks in an efficient manner for a large number of tasks submitted over a long period of time. This period has been chosen as a 50 day simulation period for this experiment.

8.17.1. Experimental Setup

- GRCs

The experiment was carried out using two sets of GRCs; the first represented thirteen QoS GRCs (Tier A) submitting up to 2200 requests over a period of 50 days. The second was carried out with thirteen FCFS users submitting the same number of requests over 50 days.

- Resources

12 Clusters are simulated within this experiment. 10 are CPU clusters containing computational resources, while 2 are Storage Clusters containing storage resources. The resource characteristics are in Table 20.

Table 20: Resource Characteristics

Type	Minimum	Maximum
CPU	2.4 GHz	3.8 GHz
RAM	128 MB	2048 MB
Storage	1GB	1024 GB

8.17.2. Metric

The metric used is the number of tasks completed per day over the period of 50 days.

8.17.3. Results

Figure 56 illustrates the results obtained from both runs.

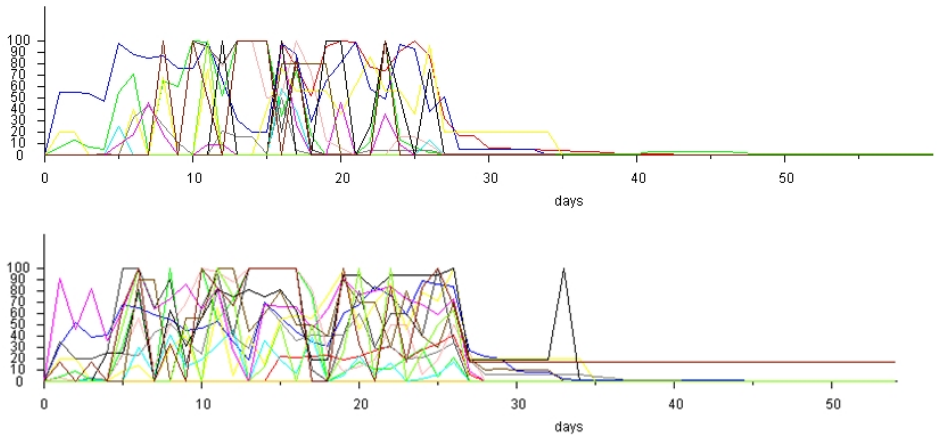


Figure 56: Comparison between BGQoS and FCFS over 50 days

The top half of Figure 56 shows the number of requests completed every day over a 50 days simulated period for BGQoS and the bottom half shows the same for FCFS. It is clear from the figure that all GRCs completed their tasks over the required period when BGQoS was used. This was not the case when using FCFS where there were tasks still running at the end of the simulated period. Moreover, we can see that task execution was more uniform and better organised under BGQoS where as task completion appears less so using FCFS.

It can therefore be conclude that BGQoS performed its resource operations, task execution and task completion driven by GRC requirements in an efficient manner illustrated in this comparison with FCFS.

8.18. Analysis of the BGQoS Operation Evaluation

This chapter has examined in detail the different relevant operations of BGQoS under different circumstances, with different parameters and within different environments ranging from small to very large. There has been an explanation of the results in each of the sections within this chapter and the importance of each is presented within this section:

Overhead: There is a measured amount of overhead related to the resource operations within BGQoS that could not be ignored. However, one of the primary goals of BGQoS is to locate the resources that meet the requirements specified by a GRC, which means that these resources must be identified, located, ranked and selected. The overhead is consequently expected and within the context of the models operation could be considered tolerable.

Constraints: The effect of Constraints has been shown clearly and the experiments justify the need to separate them from regular QoS parameters. They are the guidelines and measures that govern which resources are to be used, by which GRCs, how they are used and when. This justifies the identification of constraints as a separate set independent of QoS requirements which may depend on the constraints given, assigned or submitted.

The number of parameters: The number of parameters has had an effect on the operation of BGQoS and has been presented in detail within this chapter. Mainly, the effect was on the overhead, the number of successful requests and the percentage of tasks completed.

In general, the successful completion of requests and tasks depends on the following:

Time constraint: Increasing the time limit set for tasks to be completed increases the possibility of successfully completing the GRC request and locating the appropriate resources. This is due to resources being released and resources being added over a period of time that may have not been available within a tight timeline.

Cost constraint: Like the Time Constraint, the Cost constraint expresses the budget available for a GRC. A larger budget allows the GRC to utilise more expensive resources that may deliver a higher level of QoS. Increasing the budget and loosening the Cost constraint increases the possibility of successfully completing GRC requests and enhancing the level of QoS delivered.

Resource population: The increase in the number of resource available, either due to more resources becoming available over a period of time or resource being added and becoming accessible, increases the probability that there exists a resource that meets the requirements submitted by the GRC.

In contrast, failure to complete requests and tasks depends on the following:

GRC population: An increase in the number of GRCs and the requests submitted by them to a resource population has resulted in an increase in resource utilisation and assignments. However, it has also resulted in an increased volume of competition between the different requests for resources and larger waiting times. In some cases, with access the same population of resources, the requests have not been granted and have been returned as failures.

Increase in the number of requested parameters: The increase in parameters reduces the number of resources with the capacity to meet the GRC request. However, within the experiments carried out we have recognised that access to resources from partner Grids and Global resources has increased the possibility of these requests being met and therefore still allowed the GRC to make specific requirements with more parameters, successfully.

Access to partner and global resources: This was not a given in some experiments where the effect of constraints was not as noticeable, or the resource population did not produce a substantial increase in the number of successful requests and completed tasks. Overall, it is worth noticing that multi-parameter requests are expected to receive a considerable amount of delay, request failures and are reliant on the Cost and Time Constraints. Once the request is accepted and a resource set is selected, however, BGQoS has produced a substantial and positive successful execution rate; this was further enhanced by the introduction of reallocation according to a pre specified tolerance ratio. Moreover, in comparison with best effort operations for a large number of submitted tasks, BGQoS has produced better

performance and resource utilisation results when compared with best effort performance and utilisation, albeit, at a high price.

Overall, the optimal operation of BGQoS seems to occur when there is a large population of GRCs and resources, efficiently tailoring to the GRC requests using the information on resources. Resource information has been successfully maintained up to date using the updating mechanism attached to the databases holding the information on resources within out model.

Finally, it is important to mention that while best effort GRCs have produced substantially larger makespans than the QoS supported GRC tiers; they have still managed to out perform traditional FCFC methods by using more resource organisation and providing a higher resource utilisation rate.

8.19. Summary

This chapter has presented the experiments that have been carried out on the components and operations of BGQoS within simulated environments. Each experiment was tailored towards testing and validating a specific aspect of BGQoS, the effects of different variables such as GRC types and the conditions for which BGQoS components perform to their potential. Chapter carries on with testing BGQoS, however, the experiment is setup to examine the operation of BGQoS as a complete model. The combination of the two chapters provides a comprehensive set of results that both tests the specifics and the whole model.

CHAPTER 9 EVALUATION OF COMPLETE OPERATION OF BGQOS

9.1. Introduction

This chapter evaluates BGQoS in the context of a holistic system experiment rather than the evaluation of component characteristics and relationships as in chapter 8. It presents a comprehensive experiment tailored to evaluate the complete operation of BGQoS and investigate its success in delivering QoS according to the operational model and employing BGQoS components and methods introduced within this thesis. The results of the experiments are analysed in order to establish the validity of BGQoS, its flexibility and application potential.

9.2. The Simulated Environment

Table 21 presents the components and parameters of the simulated environment. The total numbers reflect those of real environments with real workloads. The selection of a large number of GRCs in relation to the GRPs and the resources they provide is to maintain a competitive environment, where resources are selected accordingly.

Two types of GRCs are represented, QoS GRCs and BE GRCs, split into a 40-60 percentage ratio. The QoS GRC within this experiment reflects the Tier A GRC we have introduced throughout this thesis. The BE GRC represents a Tier C GRC we have introduced throughout this thesis. Tier B has not been represented as this experiment is designed under the assumption that that if a full set of QoS can be provided then a reduced set can also be provided. This illustrates a better suited experiment for a clearer picture in terms of results between a GRC with QoS capabilities and a GRC without any. GRPs are located inside their respective Grids and are related to them. Communication occurs by registering their resources, and resource information is stored accordingly.

Table 21: Experimental Setup

Total number of accessible Grids	**0 - 500**
Total number of resource in each Grid	50
Bandwidth	100 Mb/s - 1 Gb/s
MIPS	2200 - 6000
RAM	256 MB - 2 GB
Task size	20 Kbytes -2 Mbytes
Total number of GRCs	600
Price per unit of time	10 - 500
GRC cost constraints	100 - 800
GRC time constraints	100 - 1500
Total number of GRPs	150

9.3. Evaluation Metrics

The evaluation metrics that have been chosen for this experiment are presented and explained in table 22. The table also includes the description and the measurement for each metric.

Table 22: Experimental Measurements

Metric	*Description*	*Formula/Measurement*
Response Time	Measures the time required for GRCs to receive a response to their execution request.	$\mathbf{Time_{response}}$
Resource Utilisation	Measures the resource utilisation.	$\mathbf{U_i = \frac{O_i^t}{EO_i^t}}$ where the U_i is the Utilisation of resource R_i over a period of time t. O_i^t is the actual output from R_i and EO_i^t is the estimated output from R_i over the same period of time t.
The percentage of successful GRC requests	Measures the efficiency of meeting GRC requests. A successfully met GRC request is that which identifies and locates the appropriate resources through BGQoS for a GRC requirement specification.	$\mathbf{\frac{Successful\ requests}{Total\ requests}}$
The percentage of successfully completed tasks	Measures the ration between successfully completed tasks to the total number of tasks submitted. The measurement classifies a successfully completed task as those tasks that have been executed to completion according to GRC requirements	$\mathbf{\frac{Successful\ tasks}{Total\ Tasks}}$

Effects of varying constraints	Measures the degree the constraints can effect the operation of BGQoS. It also identifies the importance of including them and the benefits of making them flexible.	**Time Constraint ,T, and overall time OT** OT must be less or equal to T, where $OT = Time_{Rop} + Time_{ft} + Time_{queue} + Time_{execution} + Time_{migration}$ and $Time_{Rop}$ the time to complete resource operations. **Cost Constraint, C, and overall Cost OC** OT must be less or equal to C, where $OC = \sum_{n=1}^{k} P(t) x\ Time_{execution}$
GRC satisfaction	Measures the level of QoS delivered in comparison to the level requested. The importance of which is identify whether BGQoS managed to maintain a sustained level of QoS throughout the execution of GRC tasks.	$\frac{\mathbf{QoS_{delivered}}}{\mathbf{QoS_{requested}}}$ The measurement is the total for all tasks and compared with the Tolerance ratio TR which submitted by the GRC

9.4. Results

The results for each of the experiments are presented in this section, including differing numbers of submitted tasks to reflect different conditions and loads.

9.4.1. Response Time

Response time has been measured with three types of ranking; i) Time minimisation ranking ii) Cost minimisation ranking and iii) Ranking according to availability. The following two figures illustrate the results obtained from the three experiments, Figure 57 presents the results when 350 tasks are submitted and Figure 58 presents the results when 1100 tasks are submitted.

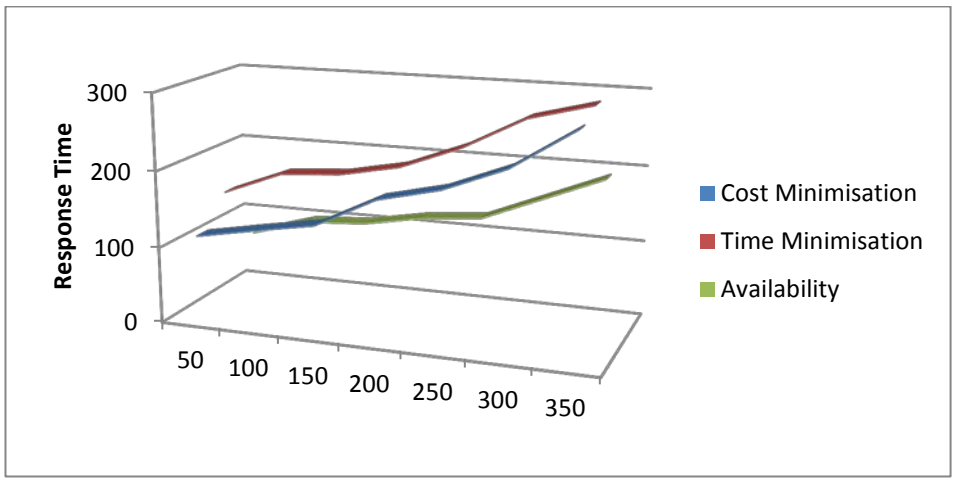


Figure 57: Response time for 350 tasks

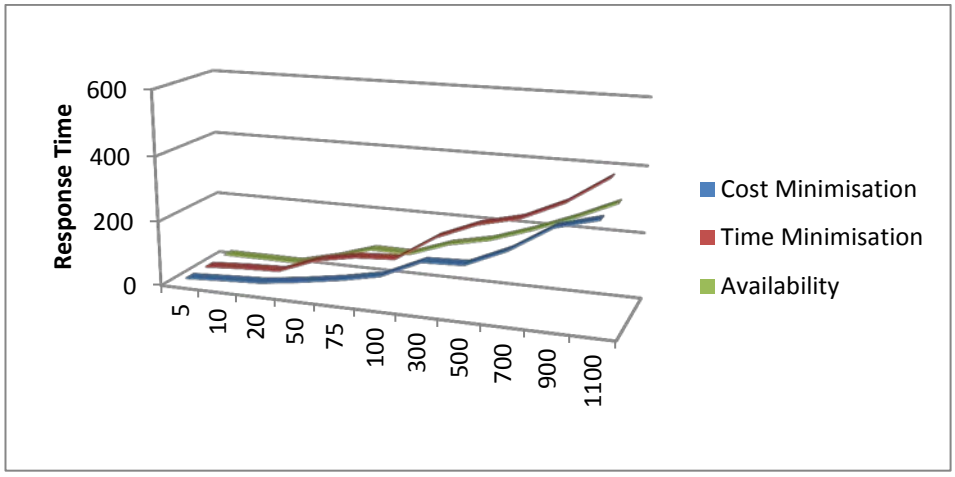


Figure 58: Response Time for 1100 tasks

9.4.2. Resource Utilisation

Resource Utilisation has been measured with three types of ranking; i) Time minimisation ranking ii) Cost minimisation ranking and iii) Ranking according to availability. The following two figures illustrate the results obtained from the three experiments, Figure 59 presents the results when 350 tasks are submitted and Figure 60 presents the results when 1100 tasks are submitted. In figure 60, it is clear that there was a significant rise in resource utilisation. This is due to a larger number of smaller tasks being carried out, therefore utilising resources to at a higher level. This is attributed to resources being able to provide a specific level of services over a shorter priod of time, rendering them available for carrying out smaller tasks which require a shorter time to execute.

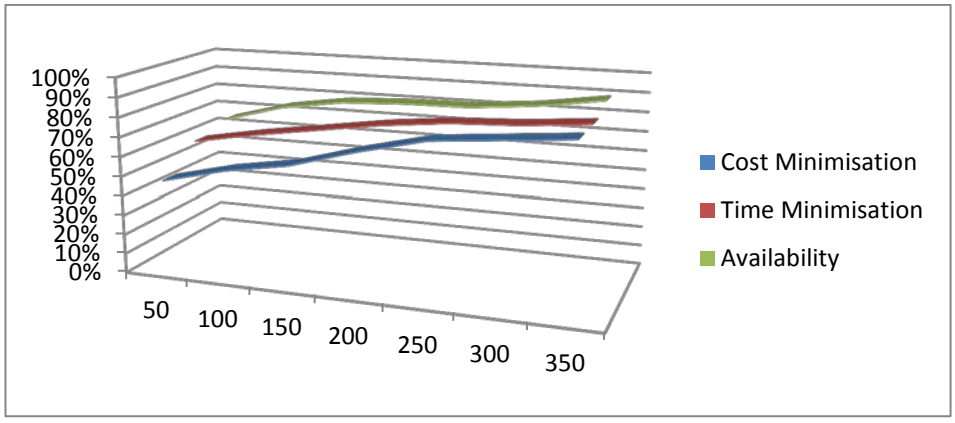


Figure 59: Resource Utilisation for 350 tasks

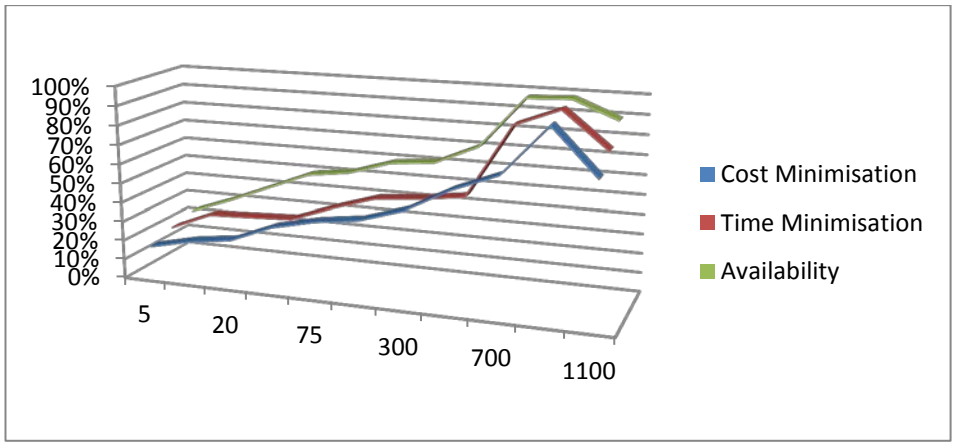


Figure 60: Resource Utilisation for 1100 tasks

9.4.3. Percentage of successful GRC requests

The percentage of successful GRC requests has been measured with three types of ranking; i) Time minimisation ranking ii) Cost minimisation ranking and iii) Ranking according to availability. The following two figures illustrate the results obtained from the three experiments, Figure 61 presents the results when 350 tasks are submitted and Figure 62 presents the results when 700 tasks are submitted.

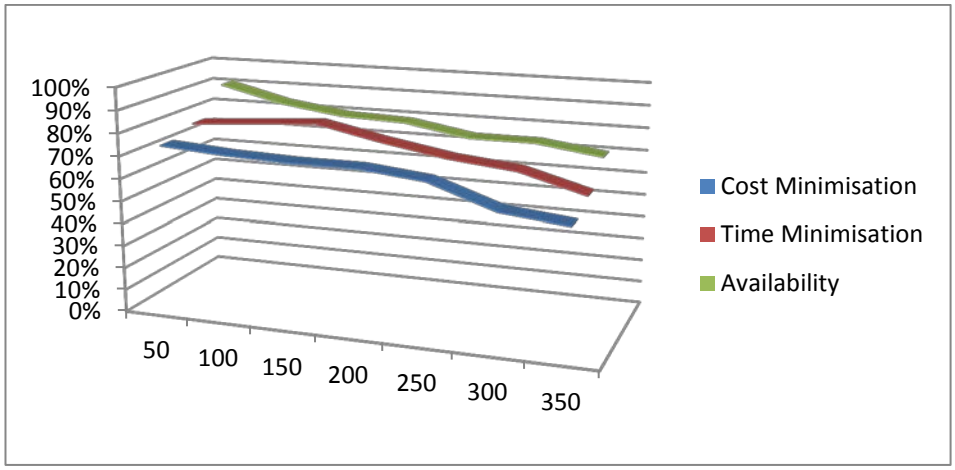


Figure 61: Percentage of successful GRC requests for 350 tasks

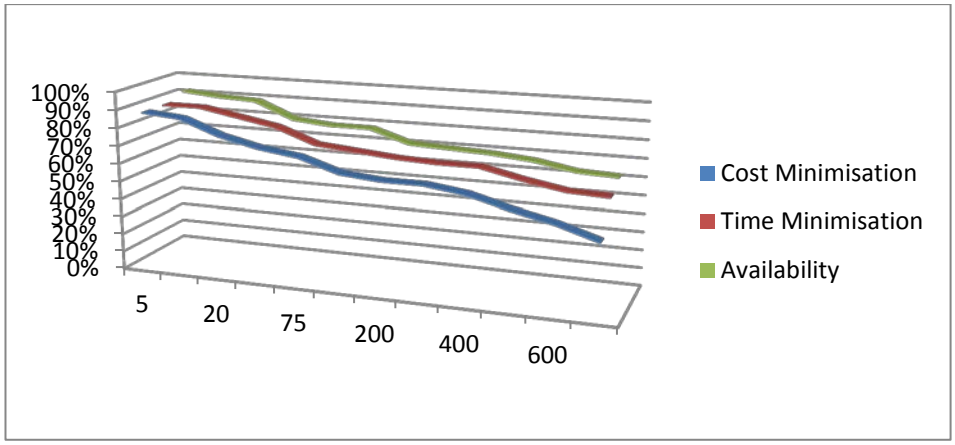


Figure 62: Percentage of successful GRC requests for 700 tasks

9.4.4. Percentage of successfully completed tasks

The percentage of successfully completed tasks has been measured with three types of ranking; i) Time minimisation ranking ii) Cost minimisation ranking and iii) Ranking according to availability. The following two figures illustrate the results obtained from the three experiments, Figure 63 presents the results when 350 tasks are submitted and Figure 64 presents the results when 1100 tasks are submitted.

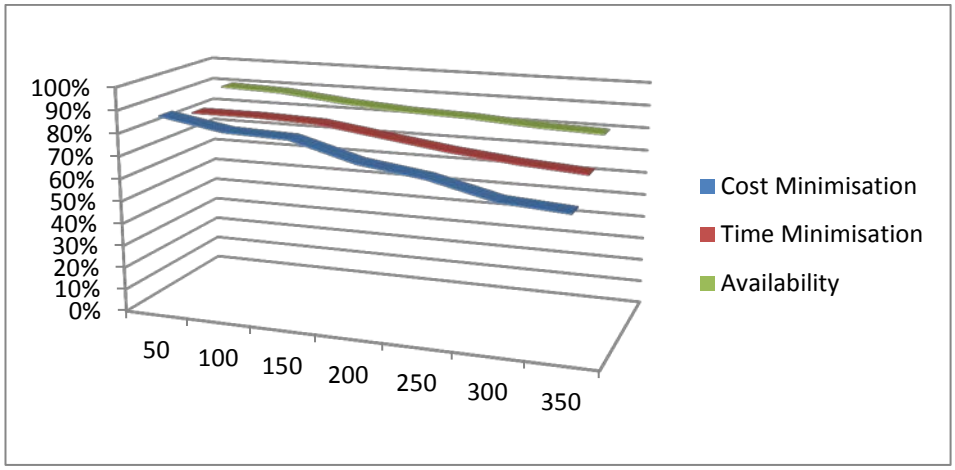


Figure 63: Percentage of successfully completed tasks for 350 tasks

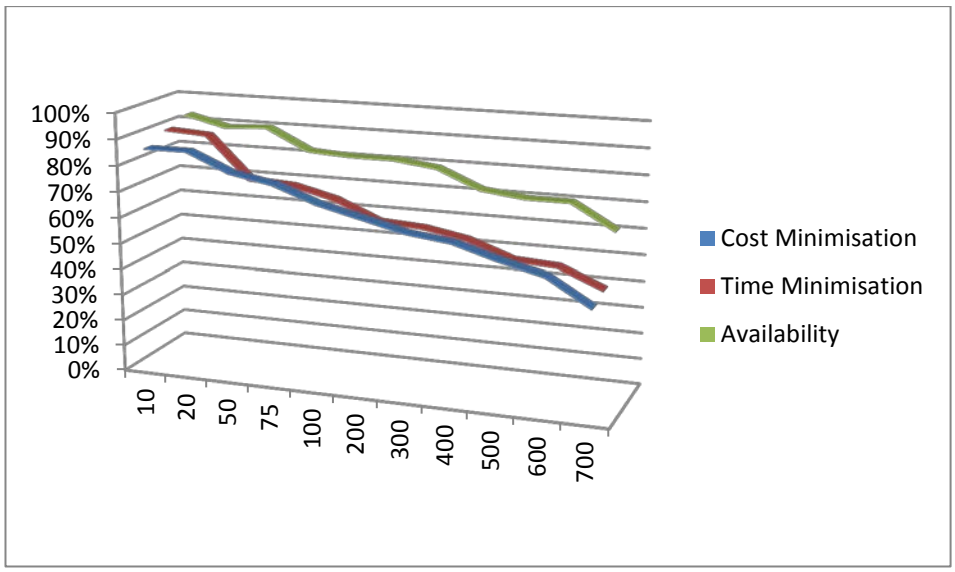


Figure 64: Percentage of successfully completed tasks for 700 tasks

9.4.5. Effect of Varying Cost and Time Constraints

This section presents the results of varying the Cost and Time Constraints. Each of which is presented within an experimental setup tailored for testing the effects of varying them.

9.4.5.1. Time Constraint

In this experiment the comparison has focused the resource utilisation, in case there is a time constraint or there is not. The time constraints have been set to 500 units, increasing by 100 units per 100 tasks submitted.

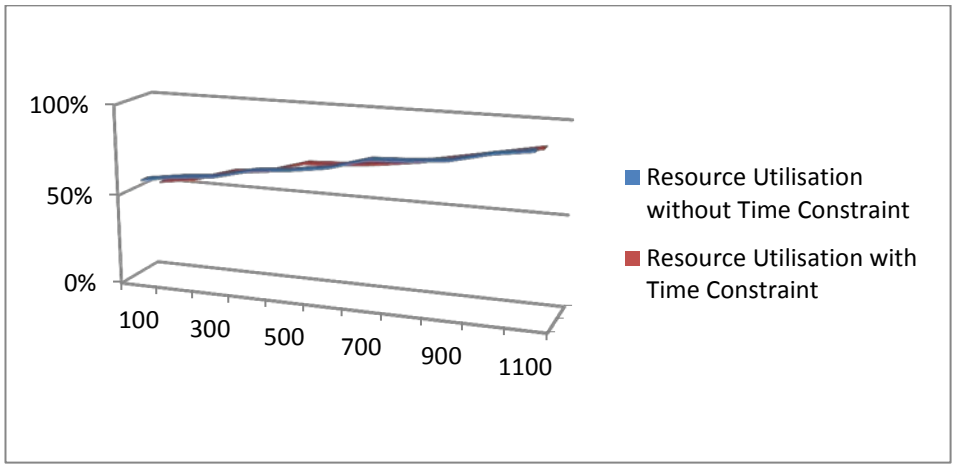


Figure 65: Effect of Time Constraint on Resource Utilisation for 1100 tasks

9.4.5.2. Cost Constraint

In this experiment we examine the percentage tasks executed successfully for 1100 tasks submitted by a GRC. In each run the C has been set to 75 increasing by 10 for each run to 750 units for 100 tasks. The time constraint has been set to 500 units, increasing by 100 units per 100 tasks submitted. Figure 66 shows the results for:

- 1 Parameter $+ T = \infty + C = (75$ to$750)$
- 2 Parameters $+ T = (500$ to $1500)$
- 3 Parameters $+ T = (500$ to $1500) + C = (75$ to $750)$

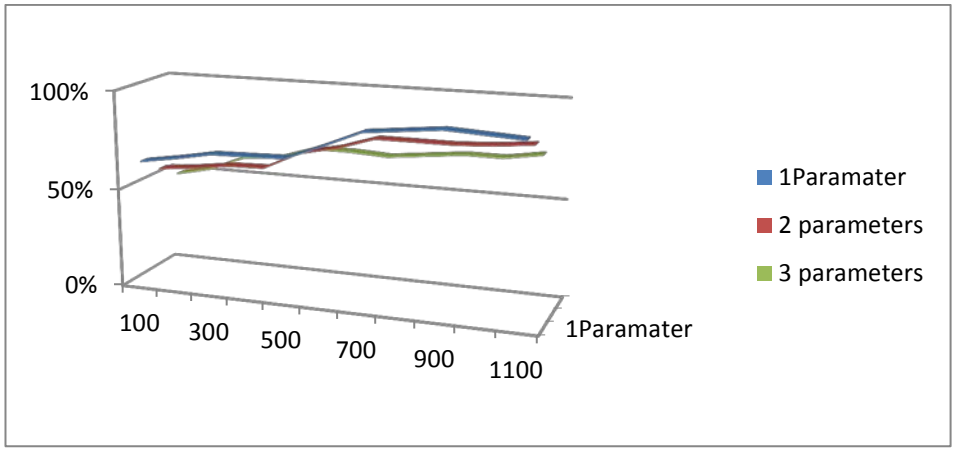


Figure 66: Tasks executed successfully on Resource Utilisation for 1100 tasks

9.4.6. GRC Satisfaction

GRC satisfaction has been measured with three types of ranking; i) Time minimisation ranking ii) Cost minimisation ranking and iii) Ranking according to availability. The following two figures illustrate the results obtained from the three experiments, Figure 67 presents the results when 350 tasks are submitted and Figure 68 presents the results when 1100 tasks are submitted. In Figure 68, the rise in GRC satisfaction is due to a larger number of tasks being completed. This is attributed to a larger number of smaller tasks being carried out, utilising resources that are available for shorter periods of time, allowing a higher level of execution return and QoS delivery.

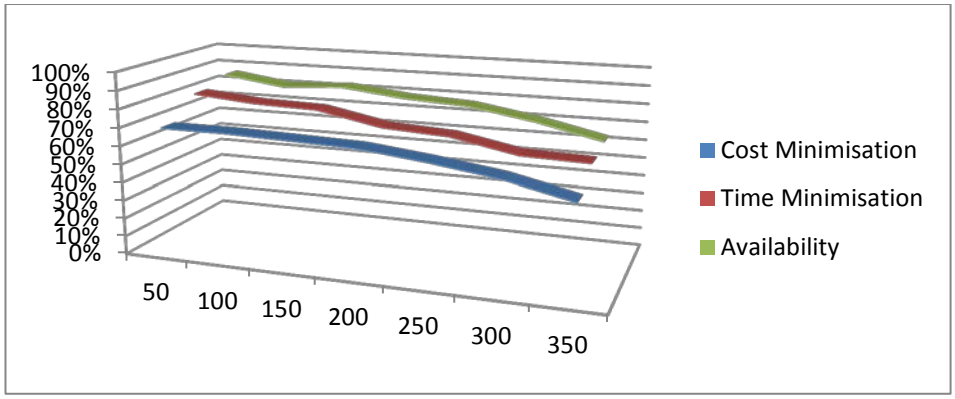


Figure 67: GRC satisfaction for 350 tasks

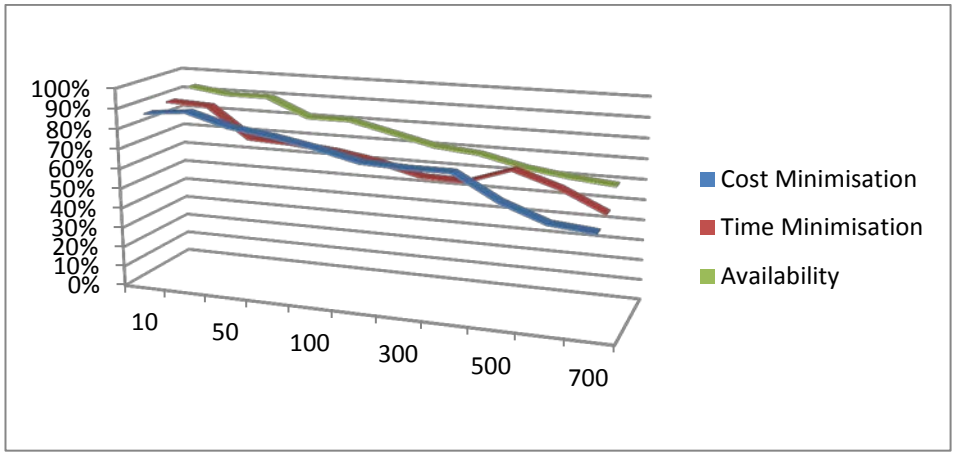


Figure 68: GRC satisfaction for 700 tasks

9.5. Analysis

The simulated environment within this section examined the different aspects of BGQoS. Overall, BGQoS successfully carried out the main operations in terms of resource operations and delivering QoS to GRCs according to their requests.

Response time increased as the number of tasks submitted increased. However, this increase is expected and still within acceptable times. Within this experiment, it peaked at just under 400 ms for 1100 requests with time minimisation ranking commanding the highest response time. However, using all three types of ranking, the results have been comparable and similar. Therefore, the ranking process has operated successfully and consistently regardless of the ranking criteria.

Resource utilisation steadily increased when 350 tasks were submitted for execution. In fact, resource utilisation achieved relatively high percentage and BGQoS's distribution of tasks has performed positively. All three types of ranking performed similarly and consistently, with resources achieving utilisation of above 80 % at the 300 tasks submitted mark.

Resource utilisation started to decrease when a large number of requests for tasks to be executed under specific parameters were submitted. The reduction of utilisation is due to multiple factors, including resources carrying a larger number of smaller tasks which may not utilise the resources to their full capacity. However, the performance was still positive, achieving a high level of utilisation throughout.

The number of successfully met requests decreased as the number of requests increased, to a consistent pool of resources. However, the decrease of almost 20 % in meeting requests compared to an increase of 50 % in tasks submitted presents a positive outcome. Moreover, successful operation of BGQoS assigning the tasks to the resources that meet the GRC requirements while facing a decrease in requests met has fulfilled its main objective.

Until the resource population is exhausted, the decrease in the number of successfully executed tasks occurred slowly. This is due to the successful allocation of tasks to resources that meet their requirements, therefore fulfilling the execution requests. However, once there is competition for the resources, when a larger number of tasks are submitted, the decrease became more significant. This is due to current tasks overrunning, resources failing, and, constraints not being met.

The effects of cost and time constraints were elaborated on within the last chapter. The experiments that have been carried out within this chapter examining their effects have been consistent with the initial examination and conclusions. In general, the larger the time available and the budget available, the larger number of tasks that are executed.

Overall, BGQoS has been successful in carrying out its core operations with a consistent and positive level of operation, achieving a high percentage of successful requests, successful executions and resource utilisation while driven by QoS parameters submitted by the GRC.

9.6. Summary

The chapter has presented an experiment that has covered different aspects of BGQoS under different conditions and parameters, concluding with the analysis of the results, from which we have shown the combining that different operation evaluated perform the required objectives and achieves a flexible high-level QoS driven model, BGQoS. Chapter 10 presents the conclusions and the future directions of the research.

CHAPTER 10: SUMMARY, CONCLUSION AND FUTURE DIRECTION

10.1. Introduction

This chapter outlines the contributions, conclusions and summary of the thesis as well as serving as the finale for it. This chapter is of two parts, the first discusses the work achieved within this thesis and the second part describes the future work and directions for the work presented.

10.2. Thesis Contributions

The thesis makes the following contributions:

- BGQoS supports a novel GRC architecture that assigns privileges to tiers instead of specific users. This both improved performance and provided a more realistic user model that could be implemented in business-oriented and commercial environments and can be easily expanded for different domains. This has been complemented by a set of tier specific interfaces and different types of requests based on different types of templates. This has also allowed BGQoS to be redesigned for specific organisations within multiple domains, making it operational and flexible.
- The description and implementation of GRCs, GRPs, Resources and SLA agreements have achieved:

- Simplicity: BGQoS allows different types of descriptions and allows them to be understandable and usable by a wide range of GRCs and allows many types of resources to de defined and described in terms of capabilities and ownerships. The XML based documents produced re machine and human readable accommodating the premise that the target GRCs cannot be assumed to have knowledge of the underlying protocols, procedures or infrastructure.

- Expressiveness: BGQoS allows GRCs to express their requirements and eliminates confusion in terms of what they can or cannot request.

- Flexibility: BGQoS accommodates different resources, GRC requirements and features. Moreover, it allows the components, including interfaces and descriptions to be tailored for a specific domain or for a specific environment without affecting the support for different types of GRCs or Resources.

- Specificity: The design of the BGQoS facilitates the identification of specific components such as the business relationship, SLOs and additional objectives or priorities between the different parties.

- High-level abstraction in which GRCs only focus on specifying the resource requirements for their applications and the QoS parameters and constraints they require without the need for technical detail specification, thus achieving a high-level QoS driven specification model.

- QoS driven resource discovery and selection has been implemented to its intended effect, in which the resources are selected according to the $QoS_{description}$ submitted by the GRC. The GRC can therefore expect that the resources selected adhere to their requirements in terms of types and meeting the level of QoS required.

- Resource advertisement has been included, within which resources can define their capabilities and characteristics and these characteristics can be matched with GRC requirements. Expressiveness of $Resource_{descriptions}$ associated with each resource have been introduced in order to clearly specify resource characteristics, properties and policies. These descriptions are stored in a specialised databases efficiently allowing access and resource information retrieval.

- Dynamic resource characteristics are supported and have been implemented efficiently where dynamic resource characteristics are updated automatically at specific time intervals ensuring that the information stored relative to each resource is based on current information and is up-to-date. This has increased the accuracy of resource discovery and resource selection operations which use this information.

- Time and cost constraints are supported allowing the GRC full control over the time limits and cost limits associated with their applications. Moreover, mechanisms have been implemented in order to carry out cost or time minimisation operations if requested by the GRC.

- A QoS model, communication organisation and agreements initialisation has also been presented and can be expanded to support multiple domains and different types of applications, GRCs and requests.

- Resource discovery is supported on three levels, local, partner and global. Driven by the GRC requests and QoS specification, resource discovery employs a search mechanism that retrieves resource information and matches them with the GRC request in order to select the most appropriate resource for selection. The selection process is carried out using a ranking process which is flexible, configurable and expandable in order to accommodate different types of GRC, resources and domains.

- The resource operations are quantified where each resource is assigned to a specific degree of matching to the requirements of the GRC, as well as a specific assigned value, called a rank.

- Feedback operations have been implemented in order to convey relevant data on resources, requests, task execution and level of QoS delivered back to the GRC.

- Agreement establishment, management and monitoring have been implemented in order to meet the GRC QoS requirements and maintain that the level promised is being delivered within specific ratio boundaries set by the GRCs.

- Recovery, reallocation and migration operations are implemented in case of violation and error, complementing the monitoring process and guaranteeing the level of QoS delivered, while hiding the complexities of migration and reallocation from the GRC. This process is carried out automatically within the execution phase of an application by BGQoS.

- Accounting and billing services have been implemented complementing the operations above and concludes the communication process between the different parties, applying costs, penalties and session status.

10.3. Conclusion

The novel contributions of the thesis can be summarised as the development of a business grid QoS system which innovatively incorporates varying QoS-based tiers of GRC and also as the extension of a simulation environment to enable experimentation in QoS support for Grids.

BGQoS, through its support mechanisms, layered architecture, components and operations has been successful in guaranteeing a sustained level of QoS for different types of QoS requirements and under different conditions within a flexible model. It has been successful in meeting the requirements of the environments for which it has been designed. The high-level design of BGQoS which employs existing technologies and new methods and techniques in resource discovery, selection and different levels of QoS support through a specific QoS model, has provided a stepping stone that could be carried forward to support the integration between the targeted domains and Grid Computing.

BGQoS has been designed to be flexible and expandable. The implementation has achieved the goals, objectives and operational requirements that have been specified, and has achieved the contributions explained in the previous section of this chapter.

A simulation environment has been used in order carry out model evaluation, with a fully functional set of components with results obtained presented in this thesis. It has showed that, while there have been a few issues with overhead, the positives significantly outweigh the negatives. The delivery of QoS in particular and the complete support for the whole process initiated by the GRC request to the billing operations have provided a platform that could be used in multiple domains and according to each organisations specifications, policies, requirements and infrastructure.

It is hoped that these contributions will be useful to future developers and researchers and thus lead to improved systems in the new virtualised environments of future business computing.

10.4. Future Work and Directions

10.4.1. Full Standardisation of Metrics and Metric Unification Support

An important area to expand on is the number of QoS requirements and the types of requirements that could be set according to a standard method of communication and metric unification between resource and service consumers and resource and service providers. This would include a dynamic pricing method relative to the level of QoS provided and adhering to different agreements set in place between different parties.

Moreover, with the reduction of overhead and improving performance there could be a case for using real-time information on resources that is updated more frequently. In addition, a method could be employed that brokers inform GRCs of resource availability that they could utilise using the real-time information on resources; this provides a more business-oriented environment and opportunity for GRPs and provides the GRC with the opportunity of utilising available resources should they wish to.

The availability of information and the direct distinction between the different types of GRCs and their locality currently allows BGQoS to carry out resource selection operations using information on global resources, however, a more frequent update on resource information on a larger set of characteristics can allow resource operations to be carried out much more effectively.

10.4.2. Expansion for Cloud Computing

As part of the future direction, we aim to tailor BGQoS for emerging Cloud Computing fields and expand them to enhance the QoS support within Cloud Computing environments. The rapid growth of these environments and their adoption by major organisations and corporations such as Amazon has added to the significance and importance of Cloud Computing and therefore, applying BGQoS would be beneficiary.

The elasticity and scalability characteristics of clouds means that there must be a solution that is capable of providing the user with the ability to specify resources and specific QoS without concern to the underlying infrastructure, keeping in line with one

of the main objectives of Clouds. BGQoS could provide that solution and can help in user and request management.

10.4.3. Testing the Operation on a Real Test-Bed

Testing and evaluation for this work has been carried *via* simulation and within simulated environments for multiple reasons that have been discussed in the previous chapter. It is therefore an objective of future work to implement the methods and strategies proposed on a real testbed.

There is a significant difference between simulated environments and real test-beds, especially in terms of resource failures and logistical and legal considerations. Therefore, real test-beds provide a bigger, sterner and more realistic challenge. Moreover, any unexpected behaviour in operation within simulated environments can be traced back easily. However, this is not the case on a real test-bed where unexpected behaviour requires more effort to trace. Overall, a real test-bed would provide for a challenging environment for testing the operations and successful implementation of BGQoS.

.

References

Aggarwal, R., Verma, K., Miller, J. and Milnor, W. (2004) 'Constraint Driven Web Service Composition in METEOR-S'. *Proceedings of the 2004 IEEE International Conference on Services Computing*. held 15-18 September, Shanghai, China, 23-30

Al-Ali, R., ShaikhAli, A., Rana, O. F. and Walker D. W. (2003) 'Supporting QoS-Based Discovery in Service-Oriented Grids'. *The 17th International Symposium on Parallel and Distributed Processing (IPDPS '03)*. held 22-26 April, Washington, DC, USA

Albodour, R. , James, A. Yaacob, N. and Godwin, A. (2008) 'QoS Requirements for a Medical Application on the Grid'. *Computer Supported Cooperative Work in Design IV*. 5236/2008, 316-330, DOI: 10.1007/978-3-540-92719-8_29, 316-330

Albodour, R. , James, A. and Yaacob, N. (2010) 'An extension of GridSim for quality of service'. *The 14th International Conference on Cooperative Work in Design*. held 14-16 April in Shanghai, China, 361 - 366

AL-Fawair, M. (2009), *A Model for Evolving Grid Computing Systems*, PhD Thesis. England: De Montfort University

Allcock, B., Bester, J., Bresnahan, J., Chervenak, A. L. and Foster, I. (2002) 'Data management and transfer in high performance computational Grid environments'. *Parallel Computing Journal* 28(5), 749–771

Allen, G., Goodale, T., Russell, M., Seidel, E. and Shalf, J. (2003) 'Classifying and enabling Grid Applications'. *Grid computing: making the global infrastructure a reality*, 601-614

Altair (2011) *Altair's Portable Batch System* [online] available from <http://www.pbsworks.com/>

Altintas, I. , Birnbaum, A. , Baldridge, K.K. , Sudholdt, W. , Miller, M. , Amoreira, C. , Potier, Y. and Ludaescher, B. edited by Herrero, P. , Perez, M.S. and Robles, V (2004). 'A Framework for the Design and Reuse of Grid Workflows'. *Scientific Applications of Grid Computing: First International Workshop, SAG 2004*. held 20-24 September in Beijing, China, Revised Selected and Invited, in series Lecture Notes in Computer Science 3, 119-132

Amazon (2011a) Amazon Elastic Compute Cloud (Amazon EC2) [online] available from <http://aws.amazon.com/ec2/>

Amazon (2011b) Amazon S3 is storage for the Internet [online] available from <http://aws.amazon.com/S3/>

Amin, K., Von Laszewski, G. and Mikler, A. R. (2004) '*Toward an Architecture for Ad Hoc Grids'. The 12th International Conference on Advanced Computing and Communications (ADCOM 2004)*. held 15-18 December in Ahmedabad Gujarat, India

Anderson, D.P. (2004) 'BOINC: a system for public-resource computing and storage'. *The Fifth IEEE/ACM International Workshop on Grid Computing*. held 8 November, ISSN: 1550-5510, Print ISBN: 0-7695-2256-4, 4-10

Andrade, R. , Oliveira, I. , Fernandes, J. M. and Cunha, J. P. (2007) *Multi-voxel Non-linear fMRI Analysis - A Grid Multi-voxel Non-linear fMRI Analysis.* In: IBERGRID, Santiago de Compostela, Spain

Andrzejak, A. , Mastroianni, C. , Fragopoulou, P , Kondo, D. , Malecot, P. , Reinefeld, A., Schütt, T. , Silaghi, G.-C. , Silva, L. M., Trunfio, P., Zeinalipour-yazti, D. and Zimeo, E. (2008) 'Grid Architectural Issues: State-of-the-art and Future Trends.' *CoreGRID White Paper Number WHP-0004*, available from <http://www.coregrid.net/>.

Apple (2011a) *Apple Science* [online] available from <http://www.apple.com/science/>

Apple (2011b) *iCloud* [online] available from < http://www.apple.com/icloud/>

Baldassari, J., Finkel, D. and Toth D. (2006) 'Slinc: A Model for Volunteer Computing'. *The 18th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2006)*. held in Dallas, USA

Barrass, T.A. , Maroney, O. , Metson, S. , Newbold, D. , Jank, W. , Garcia-Abia, P. , Hernández , J. M. , Afaq, A. , Ernst, M. , Fisk, I. , Wu, Y. , Charlot, C. , Semeniouk, I. , Bonacorsi, D. , Fanfani, A. , Grandi, C. , DeFilippis, N. , Rabbertz, K. , Rehn, J. , Tuura, L. and Wildish T. (2004) 'Software Agents in Data and Workflow Management'. *Computing in High Energy and Nuclear Physics (CHEP 2004)*. held in Interlaken, Switzerland

BEinGRID (2011) *Business Experiments in Grid* [online] available from <http://www.beinGrid.eu/>

Berman,F., Wolski, R., Casanova, H. , Cirne, W. , Dail, H. , Faerman, M. , Figueira, S. , Hayes, J. , Obertelli, G. , Schopf, J. , Shao, G. , Smallen, S. , Spring, N. , Su, A. and Zagorodnov, D. (2003) 'Adaptive computing on the Grid using AppLeS'. *Parallel and Distributed Systems* 14 (4), 369 – 382

Bobroff, N. , Dasgupta, G. , Fong, L. , Liu, Y. , Viswanathan, B. , Benedetti, F. and Wagne, J. (2008) 'A distributed job scheduling and flow management system'. *ACM SIGOPS Operating Systems Review* 42 (1), 63-70

Brandic, I. and Dustdar, S. (2011) Grid vs. Cloud - A Technology Comparison. It. *Information Technology* Issue 4

Brandich, I., Venugopal, S., Mattress, M. and Bunya R. (2008) 'Towards a Meta-Negotiation Architecture for SLA-Aware Grid Service, Workshop on Service-Oriented Engineering and Optimizations'. *The International Conference on High Performance Computing 2008 (HiPC2008)*. held 17 - 20 December in Bangalore, India

Britton, D., Doyle, A., Lloyd S. (2005) 'A Grid for Particle Physics - managing the unmanageable'. *The UK e-Science All Hands Conference*, 1073 – 1077

Brock, M., Goscinski, A. (2010) 'Grids vs. Clouds'. *The 5th International Conference on Future Information Technology*. held in Busan, S. Korea, 1-6

Buccafurri, F. , Me, P.D. , Fudging, M.G. , Fernery, R. , Goy, A. , Lax, G. , Lops, P. , Modifier, S. , Penrice, B. , Red avid, D. , Demerara, G. and Ursine, D. (2008) 'Analysis of QoS in cooperative service for real time applications'. *Data and Knowledge Engineering* 67(3), 463-484

Buscemi M. and Montanari U. (2011) 'QoS negotiation in service composition'. *Journal of Logic and Algebraic Programming* 80 (1), 13-24

Buyya, R., Abramson, D. and Giddy, J. (2000) 'Nimrod/G: An Architecture For A Resource Management And Scheduling System In A Global Computational Grid'. *The 4th international conference for High Performance Computing in the Asia-Pacific Region'*. held 14-17, Beijing , China, 283-289

Buyya, R. and Bubendorfer, K. (2009) 'The Nimrod/G Grid resource Broker for Economics-Based Scheduling, Market-Oriented Grid and Utility Computing'. *In Market-Oriented Grid and Utility Computing* : Wiley & Sons, Inc.

Buyya, R. and Murshed M. (2002) 'GridSim: a toolkit for the modelling and simulation of distributed resource management and scheduling for Grid computing'. *Concurrency and computation: practice and experience* 14 (13), 1175–1220

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J. and Brandic, I. (2009)' Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility', *Future Generation Computer Systems* 25(6), 599-616

Caminero, A., Rana, O. , Caminero, B. and Carrión, C. (2011) 'Network-aware heuristics for inter-domain meta-scheduling in Grids'. *Journal of Computer and System Sciences* 77 (2), 262-281

Cao, J., Jarvis, S. A., Saini, S. A. and Nudd, G. R. (2003) 'GridFlow: Workflow Management for Grid Computing'. *The 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*. held 12-15 May, Tokyo, Japan

Cappello, F., Djilali, S., Fedak, G., Herault, T., Magniette, F., Néri, V. and Lodygensky, O. (2005) 'Computing on Large Scale Distributed Systems: Xtremweb Architecture, Programming Models, Security, Tests and Convergence with Grid'. *Future Generation Computer Science* 21 (3), 417-437

Casanova, H., Legrand, A. and Quinson, M. (2008) 'SimGrid: A Generic Framework for Large-Scale Distributed Experiments'. *Computer Modeling and Simulation, 2008. UKSIM 2008*. held 1-3 April in Cambridge, UK, 26 - 131

Chang, R. , Lin, C. and Chen, J. (2011) 'Selecting the most fitting resource for task execution'. *Future Generation Computer Systems* 27 (2), 227-231

CERN (2011) *The ATLAS Experiment, CERN* [online] available from <http://atlas.ch/>

Chien, A., Calder, B., Elbert, S. and Bhatia, K. (2003) 'Entropia: Architecture and Performance of an Enterprise Desktop Grid System'. *Journal of Parallel and Distributed Computing* 63 (5), 597-610

Chervenak, A., Foster, I., Kesselman, C., Salisbury, C. and Tuecke, S. (1999). 'The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets', Journal of Network and Computer Applications 23 (3), 187-200

China Grid (2003) *IBM and China's Ministry of Education Launch 'China Grid'* [online] available from <http://www03.ibm.com/press/us/en/pressrelease/6134.wss>

Churches, D. , Gombas, G. , Harrison, A. , Maassen, J. , Robinson, C. , Shields, M. , Taylor, I. and Wang I. (2006) 'Programming Scientific and Distributed Workflow with Triana Service', *Concurrency and Computation: Practice & Experience* 18(10), 1021-1037

Cisco (2011) *Internetworking Technology Handbook (Quality of Service)* [online] available from <http://www.cisco.com/en/US/docs/internetworking/technology/handbook/QoS.html>

CLOUDS Lab (2010) *GridSim: A Toolkit for Modelling and Simulating Grid Computing Environments GridSim Toolkit 5.0* [online] available from <http://www.cloudbus.org/>

Cloud Computing and Distributed Systems (CLOUDS) Laboratory (2011) [online] available from < http://www.cloudbus.org/broker/>

CMS Project, (2005) *The Compact Muon Solenoid, an Experiment for the Large Hadron Collider at CERN* [online] available from <http://cms.cern.ch/>

Condor® Project [online] available from <http://www.cs.wisc.edu/condor/>

CoreGIRD (2008) *The CoreGRID Network of Excellence (NoE)* [online] available from < http://www.coregrid.net/>

Costa, G. Da , Dikaiakos, M. D. and Orlando, S. (2007) 'Nine Months in the Life of EGEE: a Look from the South'*. The 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. held in Washington, DC, USA, 281-287

Crawford, C., Dias, D., Lyengar, A., Novaes, M. and Zhang, L. (2003) *Commercial Applications of Grid Computing.* Published in: RC22702 [online] available from < http://domino.watson.ibm.com/library/cyberdig.nsf/1e4115aea78b6e7c85256b360066f0d4/6d8b284209bcd48b85256cb600731d4c?OpenDocument>

Curescu, C. and Tehrani, S. (2005) 'Time-aware Utility-based Resource Allocation in Wireless Networks'. *IEEE Transactions on Parallel and Distributed Systems* 16(7), 624-636

Czajkowski, K., Fitzgerald, S., Foster, I. and Kesselman, C. (2001) 'Grid information service for distributed resource sharing*'. The 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01)*. IEEE Computer Society, Washington, 181-184

Czajkowski, K. , Foster, I. , Karonis, N. , Kesselman, C. , Martin, S. , Smith, W. and Tuecke, S. (1998) 'Resource Management Architecture for Metacomputing Systems'. *IPPS/SPDP: Workshop 121 on Task Scheduling Strategies for Parallel Processing*. held 30 March- 3 April, 62–82

Czajkowski, K. , Foster, I. , Karonis, N. , Martin, S. , Smith, W. and Tuecke, S. (1998) 'A resource Management Architecture for Metacomputing Systems'. *The Workshop on Job Scheduling Strategies for Parallel Processing*, Lecture Notes in Computer Science 1459, 62–82.

Czajkowski, K., Foster, I., Kesselman, C., Sander, V. and Tuecke, S. (2002) edited by Feitelson, D. G. L. Rudolph, and Schwiegelshohn U. 'SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating resource Management in Distributed Systems'. *The 8th International Workshop on Task Scheduling Strategies for Parallel Processing (JSSPP '02)*. held in London, UK, 153-183

D-Grid (2005) *The German Grid Initiative* [online] available from <www.d-grid.de/>

Dabrowski C. (2009) 'Reliability in Grid computing systems' Concurrency and Computation: Practice & Experience'. *Concurrency and Computation: Practice and Experience Special Issue* 21 (8), 927–959

DataGRID (2004) *The EU DataGrid project's Data Management Work Package* [online] available from <http://edg-wp2.web.cern.ch/edg-wp2/optimization/optorsim.html>

Deelman, E. , Singh, G. , Su, M. , Blythe, J. , Gil, Y. , Kesselman, C. , Mehta, G. , Vahi, K. , Berriman, G. B. , Good, J. , Laity, A. , Jacob, J. C. and Katz, D. S. (2005) 'Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Journal of Scientific Programmig* 13(3), 219-237

Deora, V., Shao, J., Gray, W. A. and Fiddian N. J. (2003) 'A quality of service management framework based on user expectations'. *The 1st International Conference on Service-Oriented Computing (ICSOC' 03)*, 104-114

Desprez, F. and Vernois, A. (2005) 'Simultaneous scheduling of replication and computation for bioinformatics applications on the Grid*'. Challenges of Large Applications in Distributed Environments.* held in Research Triangle Park, NC, USA, ISBN: 0-7803-9043-1

Dias de Assun, M., Buyya R., and Venugopal S. (2008) 'InterGrid: a case for internetworking islands of Grids'. *Concurrency and Computation: Practice and Experience* 20 (8), 997-1024

Distributed and Parallel Systems Group, University of Innsbruck (2010) [online] available from <http://dps.uibk.ac.at/projects/askalon/>

Distributed Systems Architecture Group, Universidad Complutense de Madrdid News [online] available from <http://www.Gridway.org/doku.php?id=start>

Dogan, A. and Özgüner, F (2006) 'Scheduling of a meta-task with QoS requirements in heterogeneous computing systems'. *Journal of Parallel and Distributed Computing* 66(2), 181-196

Dong, F. and Akl S. G. (2006) 'Scheduling Algorithms for Grid Computing: State of the Art and Open Problems' *Technical Report No. 2006-504 Scheduling Algorithms for Grid Computing: State of the Art and Open Problems*. Report No. 2006-504

Elmroth, E., and Tordsson, J. (2005) 'A Grid resource Broker Supporting Advance Reservations and Benchmark-based resource Selection'. *State-of-the-art in Scientific Computing* 3732, 1077–1085

EGI (2011) *European Grid Infrastructure* [online] available from <http://www.egi.eu/>

EGEE (2008) *An EGEE comparative study: grids and clouds-evolution or revolution?* [online] available from < http://www.informatik.hs-mannheim.de/~baun/SEM0910/Quellen/EGEE-Grid-Cloud-v1_2.pdf>

EGEE (2010) *The Enabling Grids for E-sciencE project* [online] available from < http://www.eu-egee.org/>

Erberich, S.G., Silverstein, J.C., Chervenak, A., Schuler, R, Nelson, M.D. and Kesselman, C. (2007*) 'Globus MEDICUS - federation of DICOM medical imaging devices into healthcare Grids'* [online] available from < http://www.ncbi.nlm.nih.gov/pubmed/17476069>

EUROGRID (2004) *Application Testbed for European GRID computing* [online] available from <http://www.eurogrid.org/>

Fahringer, T., Prodan, R., Duan, R., Nerieri, F., Podlipnig, S., Qin, J., Siddiqui, M., Truong, H.. , Villazon, A. and Wieczorek, M. (2005) 'ASKALON: A Grid Application Development and Computing Environment'. *The 6th International Workshop on Grid Computing (Grid 2005)*

FinGrid (2007) *FinGrid - Financial Business Grid* [online] available from <http://www.finGrid.de/>

Fölling, A., Grimme, C., Lepping, J. and Papaspyrou A. (2010) 'The Gain of resource Delegation in Distributed Computing Environments'. *The Proceedings of the 15th Workshop on Task Scheduling Strategies for Parallel Processing (2010)'.* held in Atlanta, USA, 77-92

Forse.com (2011) *The platform for the social interprise* [online] available from < http://www.force.com/>

Foster I. (2002) 'What is the Grid? - a three point checklist'. *GRIDtoday* 1 (6), 22–25.

Foster, I. (2008) *Cloud, Grid, what's in a name?'* [online] available from <http://ianfoster.typepad.com/blog/2008/08/cloud-grid-what.html>

Foster, I., Kesselman, C., Lee, C., Lindell, R., Nahrstedt, K. and Roy, A. (1999) 'A distributed resource management architecture that supports advance reservations and co-allocation'. *The International Workshop on Quality of Service*. held 31 May – 03 June, London, UK, 27 – 36

Foster, I., Kesselman, C. and Tuecke S. (2001) 'The Anatomy of the Grid: Enabling Scalable Virtual Organizations'. *International Journal of High Performance Computing Applications* 15 (3), 200-222

Foster I., Kesselman C., Nick J., Tuecke S. (2003) 'The Physiology of the Grid: An Open Grid Service Architecture for Distributed Systems Integration'. *Journal: Computer Graphics Forum - CGF*

Foster, I., Zhao, Y., Raicu, I. and Lu, S. (2008) 'Cloud Computing and Grid Computing 360-Degree Compared'. *IEEE Grid Computing Environments (GCE08)*. held in Austin, USA, 1-10

Frost & Sullivan [online] available from <http://www.frost.com>

Fudzee, M. and Abawajy, J. (2011) 'QoS-based adaptation service selection broker'. *Future Generation Computer Systems* 27(3), 256-264

Gallard, J., Lèbre, A., Goga, O and Morin, C. (2008) 'VMdeploy: Improving Best-Effort Task Management in Grid'5000', institut national de recherche en informatique et en automatique

GangSim (2006) *GangSim: A Simulator for Grid Scheduling Studies with support for uSLAs* [online] available from <http://people.cs.uchicago.edu/~cldumitr/GangSim/>

GÉANT (2011) *The pan-European data network dedicated to the research and education community* [online] available from < http://www.geant.net/ >

Ghosh, S., Rajkumar, R., Hansen, J. and Lehoczky, J. (2003) 'Scalable resource allocation for multi-processor QoS optimization'. *The 23rd International Conference on Distributed Computing Systems*. held 19-22 May, 174–183

Global Grid Forum (GGF) (2003a) *Distributed resource Management Application API Working Group (DRMAA-WG)* [online] available from <http://www.drmaa.org/>

Global Grid Forum (2003b) *GRAAP-WG* [online] available from < http://forge.gridforum.org/projects/graap-wg>

GLOBUS (2011) *The globus alliance* [online] available from <http://www.globus.org/toolkit>

Golconda, K.S. and Ozguner, F. (2004) 'A comparison of static QoS based scheduling heuristics for a meta-task with multiple QoS dimensions in heterogeneous computing'. *The 18th International Parallel and Distributed Processing Symposium*. held 26-30 April.

Google (2011) *Google docs* [online] available from <http://docs.google.com>

GRAM (2011) *Globus resource Allocation Manager (GRAM)* [online] available from <http://www.globus.org/gram/>

Grid'5000 [online] available from <https://www.Grid5000.fr>

GridEcon (2007) *Grid Economics and Business Models* [online] available from <http://www.Gridecon.eu>

GridPhyN (2006) *Grid Physics Network* [online] available from <http://www.griphyn.org/index.php>

GridPP (2011) *UK Grid for Particle Physics* [online] available from <http://www.gridpp.ac.uk/>

Grid Workflow Archive [online] available from <http://gwa.ewi.tudelft.nl/>

GSSIM (2009) *Grid Scheduling Simulator* [online] available from<http://www.gssim.org/>

Guan, Y., Ghose, A. K. and Lu Z., (2006) 'Using constraint hierarchies to support QoS-guided service composition*'. The IEEE International Conference on Web Service (ICWS'06)*. held in Chicago, USA, 743–752

Hasselmeyer, P., Mersch, H., Kolle,r B. , Quyen, H.-N. and Schubert Lutz, W. (2007) 'Implementing an SLA Negotiation Framework'. *The eChallenges e2007 Conference*. held in Hague, Netherlands, 154-161

He, X., Sun, X. and Laszewski, G. (2003) 'A QoS Guided Scheduling Algorithm for Grid Computing'. *The International Workshop on Grid and Cooperative Computing (GCC02)*, 442-450

Hoheisel, A. (2004) 'User Tools and Languages for Graph-based Grid Workflows'. *Special Issue of Concurrency and Computation: Practice and Experience* 18 (10), 1101-1113

Hovestadt, M., Kao, O. , Keller, A. , Streit, A. edited by Feitelson, D. G. , Rudolph, L. and Schwiegelshohn, U. (2003) 'Scheduling in HPC resource management systems: Queuing vs. planning'. *The 9th International Workshop, (JSSPP 2003)*, 1–20

Huang, R., Casanova, H. and Chien, A.A. (2006) 'Using Virtual Grids to Simplify Application Scheduling*'. Proceedings of the 20th international conference on Parallel and distributed processing*. held in Rhodes Island, Greece

Iamnitchi, A. and Foster, I. (2001) 'On fully decentralized resource discovery in Grid environments'. *Proceedings of the Second International Workshop on Grid Computing (GRID '01).* held in London, UK, 51-62

IBM (2007) *IBM Grid Storage* [online] available from <http://www.IBM.com/Grids>

ICT (2011) *Information and Communication Technologies* [online] available from <http://cordis.europa.eu/fp7/ict/>

In, J.U., Avery, P., Cavanaugh, R. and Sanjay, R. (2004) 'Policy based scheduling for simple quality of service in Grid computing'. *The 18th international Parallel and Distributed Processing Symposium*. held in Santa Fe, New Mexico 26-April 30

IT-Tude E (2011) [online] available from < http://www.it-tude.com/>

Jacob B. (2003) 'Grid computing: What are the key components?' *Redbooks Project, IBM* [online] available from < http://www.redbooks.ibm.com>

Jacq, N. , Salzemann, J. , Jacq, F. , Legr´e, Y. , Medernach, E. , Montagnat, J. , Maaß, A. , Reichstadt, M. , Schwichtenberg, H. , Sridhar, M. , Kasam, V. , Zimmermann, M. , Hofmann, M. and Breton, V. (2008). 'Grid-enabled virtual screening against malaria', *Journal of Grid Computing* 6(1), 29–43

Jeffery K. G. (2007) 'Next Generation GRIDs for environmental science' *Environmental Modelling and Software* 22(3), 281-287

Jeffery K.G., Neidecker-Lutz,B., Schubert, L., and Tsakali, K. (2010) 'Cloud Computing: The next big thing?' , ERCIM News, October [online] available from <http://ercim-news.ercim.eu/en83/keynote-cloud-computing-the-next-big-thing>

Jhs, S., Merzky, A., Fox, G. (2009) 'Using clouds to provide gridswith higher levels ofabstraction and explicitsupport for usage modes', *Concurrency and Computation: Practice and Experience* 21 (8), 1087-1108

Junqueira, F.P. and Marzullo, K. (2005) 'The Virtue of Dependent Failures in Multi-site Systems,' HotDep2005

Kacsuk, P. and Sipos G. (2006) 'Multi-Grid, Multi-User Workflows in the P-GRADE Grid Portal'. *Journal of Grid Computing* 3(3-4), 221-238

Katchabaw, M.J., Lutfiyya, H.L. and Bauer, M.A. (1998) *A quality of service management testbed.* 'The Proceedings of the IEEE Third International Workshop on Systems Management'. held in Washington, DC, USA

Keahey K. and Motawi K. (2003) *Technical Memorandum ANL/MCS-TM-262*

Kenyon C. and Cheliotis G. edited by Nabrzyski, J. , Schopf, J. M. and J. Weglarz (2004) 'Grid resource commercialization: economic engineering and delivery scenarios'. *Grid resource management*, 465-478

Kertesz, A., Kacsuk, P. K., Rodero, I., Guim, F. and Corbalan, J. (2007) 'Meta-Brokering requirements and research directions in state-of-the-art Grid resource Management', *Technical Report, CoreGRID*

Kim, J. , Shivle, S. , Siegel, H. , Maciejewski, A. , Braun, T. D. , Schneider, M. , Tideman, S. , Chitta, R. , Dilmaghani, R. B. , Joshi, R. , Kaul, A. , Sharma, A. , Sripada, S. , Vangari, P. and Yellampalli, Siva S. (2007) 'Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment'. *Journal of Parallel and Distributed Computing* 67 (2), 154-169

Krauter, K., Buyya, R. and Maheswaran M. (2002) 'A Taxonomy and Survey of Grid resource Management Systems for Distributed Computing'. *Software: Practice and Experience (SPE)* 32(2), 135-164

Krishnamurthy, S., Sanders, W. H. and Cukier, M. (2001) 'A Dynamic Replica Selection Algorithm for Tolerating Timing Faults'. *Proceedings of the 2001 International Conference on Dependable Systems and Networks (DSN '01)*, 107-116

LCG (2011) *Worldwide LHC Computing Grid* [online] available from http://lcg.web.cern.ch/LCG/

Ledlie J., Shneidman J., Seltzer M., and Huth J. (2003) 'Scooped Again'. *The 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*. held in Berkeley, USA

Legrand, A., Marchal, L. and Supérieuredelyon, É. (2003) 'Scheduling distributed applications: the SimGrid simulation model'. *The Third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)*, 138-145

LHC (2011) *The LHC (Large Hadron Collider)* [online] available from <http://www.lhc.ac.uk/>

Li C., Li L. and Lu Z. (2005) 'Utility driven dynamic resource allocation using competitive markets in computational Grid'. *Advances in Engineering Software* 36 (6), 425-434

Li H., Cheng C., Chau K. (2007) 'Parallel resource Co-Allocation for the Computational Grid'. *Computer Languages, Systems & Structures 33* (1), 1-10

Liu, K. , Jin, H. , Chen, J. , Liu, X. , Yuan, D. and Yang, Y (2010) 'A Compromised-Time-Cost Scheduling Algorithm in SwinDeW-C for Instance-Intensive Cost-Constrained Workflows on a Cloud Computing Platform'. *International Journal of High Performance Computing Applications* 24 (4), 445-456

Lovas, R. , Dzsa, G. , Kacsuk, P. , Podhorszki, N. and Drtos, D. (2004) 'Workflow Support for Complex Grid Applications: Integrated and Portal Solutions'. *The 2nd European Across Grids Conference*. held 28-30 January in Nicosia, Cyprus

Ma, T., Yan, Q., Li,u W., Guan, D., Lee, S. (2011) 'Grid Task Scheduling: Algorithm Review'. *IETE Tech Journal* 28 (2)

MANET Charter [online] available from <http://www.ietf.org/html.charters/manetcharter.html>

MediGrid (2005) [online] available from <http://www.medigrid.de/index_en.html>

Menasce, D.A. and Casalicchio,E. (2004) 'A Framework for resource allocation in Grid computing'. *The The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS '04)*. held in Washington, DC, USA

MicroGrid (2003) *MicroGrid: Online Simulation Tools for Grids, Distributed Systems and the Internet* [online] available from <http://www-csag.ucsd.edu/projects/Grid/microGrid.html>

Microsoft (2011) *Windows Azure Platform/Microsoft Cloud Services* [online] available from < http://www.microsoft.com/windowsazure/>

Middleton, S. E., Surridge, M., Benkner, S. and Engelbrecht, G. (2007) 'Quality of Service Negotiation for Commercial Medical Grid Service'. *Journal of grid computing* 5 (4), 429-447

Middleton, S. E., Surridge, M., Nasser, B. I. and Yang, X. (2009) *Bipartite electronic SLA as a business model to support cross-organization load management of real-time online applications*. 'Real Time Online Interactive Applications on the Grid'

Munir, E., Li, J. and Shi, S. (2007) 'QoS sufferage heuristic for independent job scheduling in Grid'.*Information Technology Journal* 6 (8), 1166-1170

myGrid (2011) [online] available from <http://www.mygrid.org.uk/>

myGrid@EBI (2002) *European Bioinformatics Institute* [online] at: <http://www.ebi.ac.uk/mygrid/>

Nam, D., Youn, C. , Lee, B. , Clifford G. and Healey J. (2004). 'QoS-Constrained resource Allocation for a Grid-Based Multiple Source Electrocardiogram Application Computational Science and Its Applications'. *Lecture Notes in Computer Science* 3043/2004, 352-359

Netto, M. and Buyya, B. (2010) 'Resource Co-Allocation in Grid Computing Environments'. *Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications*

NGG (2011) *Next generation computing expert group* [online] available from< http://cordis.europa.eu/ist/>

NGS (2011) *The UK National Grid Service* [online] available from http://www.grid-support.ac.uk/

Ninf: A Global Computing Infrastructure (2007*) Bricks: A Performance Evaluation System for Grid Computing Scheduling Algorithms* [Online] available from <http://ninf.apGrid.org/bricks/>

NOGRID FPM [online] available from <http://www.noGrid.com/>

Nou, R. and Toerres, J (2009) 'Heterogeneous QoS resource Manager with Prediction'. *The Fifth International Conference on Autonomic and Autonomous Systems*. held 20-25 April in Valencia, Spain

Oinn, T., Addis, M. J., Ferris, J., Marvin, D. J., Greenwood, M., Carver, T. Wipat, A. and Li, P. (2004) 'Taverna, lessons in creating a workflow environment for the life sciences'. *Concurrency and Computation: Practice & Experience - Workflow in Grid Systems* 18 (10), 1067- 1100

Papazoglou, M., Traverso, P, Dustdar, S. and Leyman, F. (2008) 'Service-Oriented Computing: A Research Roadmap'. *International Journal of Cooperative Information Systems* 17 (2), 223–255

Pathak, J., Treadwell, J., Kumar, R., Vitale, P. and Fraticelli, F. (2005)' A Framework for Dynamic Resource Management On The Grid'. *Technical reports: HPL-2005-153* [online] available from < http://www.hpl.hp.com/techreports/2005/HPL-2005-153.html>

Pautasso, C. (2004) 'JOpera Visual Composition of Grid Service'. *ERCIM News No. 59, October 2004*

Pinedo, L. (2005) 'Planning and scheduling in manufacturing and service', New York: Springer Science+Business Media

Plale, B., Gannon, D., Reed, D.A., Graves, S.J., Droegemeier, K., Wilhelmson, B. and Ramamurthy M. (2005) 'Towards Dynamically Adaptive Weather Analysis and Forecasting in LEAD'. *The 5th International Conference on Computational Science.* held in Atlanta, GA, USA

Platform (2009) *Platform Load sharing facility* [online] available from < http://www.platform.com/workload-management/high-performance-computing/lp>

Quan, D. and Yang, L. (2009) 'Improving the Quality of Mapping Solutions in the System Supporting SLA-Based Workflows with Parallel Processing Technology.' *CISIS 2009*, 445-450

Ren, K., Xiao, N., Song, J., Zhang, W. and Chen, T. (2006) 'A Semantic-based Meteorology Grid Service Registry, Discovery and Composition Model'. *The Second International Conference on Semantics, Knowledge, and Grid (SKG'06)*. held 1-3 November, Guilin, Guangxi, China, ISBN: 0-7695-2673-X

Rikitake, K. and Rikitake, K. (2005) 'A Study of DNS Transport Protocol for Improving the Reliability'. *Technical Report*

Ropars, T. , Jeanvoine, E. and Morin, C. (2006) 'Providing QoS in a Grid Application Monitoring Service'. *inria-00121059, version 3*

Rappa, M. (2004) 'The utility business model and the future of computing service'. *IBM Systems Journal* 43 (1), 32-42

Ross R. and Westerman G. (2004) 'Preparing for utility computing: The role of IT architecture and relationship management'. *IBM Systems Journal* 43 (1), 5-19

Rowstron A. and Druschel P. (2005) 'Pastry: Scalable, Distributed Object Location and Routing for Large Scale Peer-to-Peer Systems'. *The 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*. held in Heidelberg, Germanyp, 329–350

Roy A. and Sander V. (2004) edited by J. Nabrzyski, J. M. Schopf, and J. Weglarz 'GARA: a uniform quality of service architecture'. *Grid resource management*, 377-394

Sakellariou, R. and Yarmolenko, V. (2008) 'Task Scheduling on the Grid: Towards SLA-Based Scheduling'. *High Performance Computing and Grids in Action* 16, 1-16

Saxena, N., Tsudik, G. and Yi, J. H. (2003) 'Admission Control in Peer-to-Peer: Design and Performance Evaluation'. *The 1st ACM workshop on Security of ad hoc and sensor networks*, 104–113

Scale Out Softwae (2011) Distributed Data Grids for the Enterprise [online] available from <http://www.scaleoutsoftware.com/>

Schmidt, R., Benkner, S., Brandich, I. and Engelbrecht, G. (2005) 'Applying a Component Model to Grid Application Service'. *The Tenth International Workshop on Component-Oriented Programming (WCOP 2005)*. held in Glasgow, Scotland

Jeffery K. G. and Neidecker-Lutz, B (2010) 'The future of CLOUD Computing'. *Report for EC CLOUD Computing Expert Group*

Selvarani, S. and Sadhasivam, D. (2010) 'Improved job-grouping based pso algorithm for task scheduling in grid computing', *International Journal of Engineering Science and Technology* 2(9), 201, 4687-4695

Shanten, J., and Andre, M., and Geoffrey, F. (2009) 'Using clouds to provide gridswith higher levels ofabstraction and explicitsupport for usage modes', *Concurrency and Computation: Pract. Exper.* 21 (8), 1087-1108

SORMA (2011) *SORMA - Self-Organizing ICT resource Management*. [online] available from <http://www.im.uni-karlsruhe.de/sorma/index.htm>

Stankiewicz, R. , Cholda, P. and Jajszczyk, A. (2011) 'QoX: What is it really?'. *Communications Magazine* 49 (4), 148 - 158

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. Chord (2001) 'A scalable peer-to-peer lookup service for internet applications'. *The 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 149-160

STFC (2011) *Science and Technology Facilities Council* [online] available from <http://www.stfc.ac.uk/>

Suß, W., Jakob, W., Quinte, A. and Stucky, K.-U. (2005) 'GORBA: A global optimising resource broker embedded in a Grid resource management system'. *The International Conference on Parallel and Distributed Computing Systems*, 19–24

Sulistio, A. , Cibej, U. , Venugopal, S., Robic, B. and Buyya R. (2007) 'A Toolkit for Modelling and Simulating Data Grids: An Extension to GridSim'. *Concurrency and Computation: Practice and Experience* 20(13), 1591 - 1609

Sun (2011) *Sun Grid Engine* [online] available from <http://wikis.sun.com/display/GridEngine/Home>

Sun, Y. , He, S. and Leu, J. (2007) 'Syndicating Web Service: A QoS and user-driven approach'. *Decision Support Systems* 43 (1), 243-255

Taher, L. and Khatib, H.E. (2005) 'A framework and QoS matchmaking algorithm for dynamic web service selection'. *The Second International Conference on Innovations in Information Technology (IIT'05)*. held in Dubai, United Arab Emirates

Takefusa, A., Nakada, H., Kudoh, T. and Tanaka, Y. (2010) 'An Advance Reservation-Based Co-allocation Algorithm for Distributed Computers and Network Bandwidth on QoS-Guaranteed Grids'. *The Proceedings of the 15th international conference on Job scheduling strategies for parallel processing*, 16-34

Taverna (2011) *An open source domain independent Workflow Management* [online] available from <http://www.taverna.org.uk/

Taylor, I. (2006) 'Triana generations, in: Scientific Workflows and Business Workflow Standards in e-Science'. *The Second IEEE International Conference on e-Science*. held in Amsterdam, Netherlands

Taylor, S., Surridge, M. and Marvin, D. (2009) 'GRIA: Grid resources for Industrial Applications'. *IEEE International Conference on Web Services*. held 6-9 July in Southampton, UK, ISBN: 0-7695-2167-3, 402-409

Thickins G. (2003) 'Utility Computing, The Next New IT Model'. *Darwin Magazine,* April

Traversat, B., Abdelaziz, M. and Pouyoul, E. (2003) 'A Loosely-Consistent DHT Rendezvous Walker.' *Sun Microsystems, Inc, Technical Report, March.*

Tserpes, K., Kyriazis, D., Menychtas, A. , Varvarigou, T.A. , Silvestri, F. and Laforenza, D. (2007) 'An Open Architecture for QoS Information in Business Grids'. *Proceedings of CoreGRID*, 37-49

UK e-Science Grid (2010) [online] available from <http://www.rcuk.ac.uk/escience/>

Unicore (2011) *Uniform Interface to Computing Resources* [online] available from <http://www.unicore.de/>

UniGrids (2006) *Uniform Interface to Grid Services* [online] available from <http://www.unigrids.org/papers.html>

Vaquero, L.M., Rodero-merino, L., Caceres, J., Lindner, M. (2009) 'A break in the clouds: Towards a cloud definition', *Computer Communication Review* 39 (1), 50-55

Venugopal, A., Buyya, R. and Winton, L. (2006)' A Grid Service Broker for Scheduling e-Science Applications on Global Data Grids', *Concurrency and Computation: Practice and Experience* 18 (6), 685-699

Venugopal, S., Xingchen, C. and Buyya, R. (2008) 'A Negotiation Mechanism for Advance resource Reservations Using the Alternate Offers Protocol'. *The 16th International Workshop on Quality of Service*. held 2-4 June in Enschede, 40-49

Wieczoreka, M., Hoheisel, A. and Prodan, R. (2009) 'Towards a general model of the multi-criteria workflow scheduling on the Grid Marek'. *Future Generation Computer Systems* 25 (3), 237-256

"WISDOM" (2005) [online] available from <http://wisdom.eu-egee.fr/>

Xia, H. , Casanova, H. and Chien, A. (1999) 'The MicroGrid: Using Online Simulation to Predict Application Performance in Diverse Grid Network Environments'. *The 2nd International Workshop on Challenges of Large Applications in Distributed Environment*. held in Washington, DC, USA

Xin, L., Xia, H. and Chien, A. (2004). 'Validating and Scaling the MicroGrid: A Scientific Instrument for Grid Dynamics.' *Journal of Grid Computing* 2 (2), 141-161

Yeo, C. , de Assunção, M. , Yu,J. , Sulistio, A. , Venugopal, S. , Placek, M. and Buyya, R. (2006) 'Utility Computing and Global Grids'. *Technical Report, GRIDS-TR-2006-7, Grid Computing and Distributed Systems Laboratory*.

Yeo, C.S., Venugopal, S. , Chu, X. and Buyya, R. (2010), 'Autonomic metered pricing for a utility computing service'. *Future Generation Computer Systems* 26 (8), 1368-1380

Yu, T. and Lin, K.J. (2005) 'Service selection algorithms for composing complex service with multiple QoS constraints'. *The Third International Conference on Service Oriented Computing (ICSOC'05)*. held in Amsterdam, the Netherlands.

Zomaya, A.Y., Ward, C. and Macey, B. (1999) 'Genetic scheduling for parallel processor systems: comparative studies and performance issues'. *Parallel and distributed Systems* 10 (8), 795 – 812

Appendix A

A.1. Introduction

The simulated environment within which BGQoS has been implemented and evaluated is an expansion of a popular simulation toolkit, GridSim. The reasons why GridSim has been chosen and details of the expansion and its importance to implement BGQoS has been explained in Chapter 7. This appendix presents a portion of this expansion, including the matchmaker and QoS Task description classes.

A.2. Matchmaker

```
import java.io.ObjectInputStream;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Collections;
import java.util.Date;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Map;
import java.util.Vector;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentLinkedQueue;
import java.util.concurrent.CopyOnWriteArrayList;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.atomic.AtomicLong;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import dsm.BGQoS.BGQoSEntity;
import dsm.BGQoS.env.TaskCenterManager;
import dsm.BGQoS.env.BGQoSMediator;
import dsm.BGQoS.env.BGQoSMessage;
import dsm.BGQoS.env.BGQoSParam;
import dsm.BGQoS.env.BGQoSPlatform;
import dsm.BGQoS.env.BGQoSToolkit;
import dsm.BGQoS.ext.TaskComparator;
import dsm.BGQoS.ext.LengthComparator;
import dsm.BGQoS.model.TaskItem;
import dsm.BGQoS.model.MMResult;
import dsm.BGQoS.model.NeighborItem;
import dsm.BGQoS.model.ResourceInfo;
import dsm.BGQoS.model.TaskInfo;
```

```
import dsm.BGQoS.storage.GlobalStorage;

/**
Class MatchMaker represents the Scheduler. It receives Tasks sent from MatchMakerController, and launches the corresponding algorithm to make the scheduling decision. Presently, implemented algorithms include FCFS, Easy Backfilling, Flexible Backfilling, EDF, EG-EDF (previously implemented) and QoS search, FCFS, Easy Backfilling, Flexible Backfilling, EDF, EG-EDF have been implemented by GridSim and GridSim extensions. TaskInfo represents a Task with dynamic information. ResourceInfo represents a resource with dynamic information.
 */

public class MatchMaker {

private String BGQoSIdentity;

private BGQoSEntity BGQoS;

/** List of separated schedules of resources */
 private LinkedList scheduleList;
 private int tempCount = 0;
 private int tempCount2 = 0;

 /** Number of already made schedulers by this MatchMaker */
   private int numOfExistingSchedules = 0;

/** Total time used for schedule generation,
    * i.e. time += Sum(clockAfterMakingSchedule - clockBeforeMakingSchedule) */
   private double totalSchedulingTime = 0.0;

/** Clock/Time before making a single schedule*/
   private double clockBeforeMakingSchedule = 0.0;

 /** Clock/Time after making a scheduling */
   private double clockAfterMakingSchedule = 0.0;

 /** QoS list of Gridlets already moved by QoS Search, EXISTING in useSchedule() */
   private LinkedList QoSGridlets = new LinkedList();

  /** Total tardiness of Tasks processed by this matchMaker; checking scheduling results */
   private double totalTaskTardiness = 0.0;

  /** incoming Task queue */
   private CopyOnWriteArrayList<String> localTaskQueue = new CopyOnWriteArrayList<String>();
   private ConcurrentHashMap<String, Double> shadowTaskQueue = new ConcurrentHashMap<String, Double>();

/** Number of Tasks waiting for scheduling decision
     It will be decrease only if the Task is already sent to aresource
```

```
     */
    private int numOfTaskWaitingForSchedule = 0;

/** Start time of the simulation; checking scheduling results */
    private double simulationStartTime = -10.0;

/** Total Task weight: LOOP all Task (numberOfCPU for execution * Task actual CPU time) */
    private double totalTaskWeight = 0.0;

/** Total time used to execute all the Tasks = execution time + I/O time + etc */
    private double totalTaskResponseTime = 0.0;

    private double totalTaskWaitingTime = 0.0;

   /** Total weighted response time
     * = (Task weight * Task response time)
     * = (Task used cpu number * Task actual cpu time * Task response time) */
    private double totalWeightedResponseTime = 0.0;

    private double totalTaskCPUTime = 0.0;

    private double totalWeightedTaskCPUTTime = 0.0;

    /** Total slowdown of Tasks = Task response time / Task actual execution time */
    private double totalSlowdown = 0.0;

    /** Total weighted slowdown
     * = (Task weight * Task slowdown)
     * = (Task used cpu number * Task actual cpu time * Task slowdown)
     */
    private double totalWeightedSlowdown = 0.0;

    /** denotes queue/schedule strategy */
    private String matchMakerPolicy = BGQoSMessage.PolicyFCFS;

    /** denotes time required to select Task*/
    String timeToSelectTaskText = "";

    /** denotes time required to add Task into queue/schedule */
    String timeToAddTaskToScheduleQueueText = "";

    // --- local Task category ---
    /** Task is considered as successful if Task.getStatus() == BGQoSMessage.SUCCESS */
    private int numOfSuccessProcessedLocalTask = 0;

    /** Task is considered as failed if:
     * (1) no resource to send to, or (2) Task.getStatus() != BGQoSMessage.SUCCESS
     */
```

```
    private int numOfFailedProcessedLocalTask = 0;

    /** Number of Tasks submitted through submitter */
    private int numOfReceivedLocalTasks = 0;


    /** Number of Tasks submitted through submitter */
    private int numOfReceivedPartnerTasks = 0;

    /** Total number of nondelayed Tasks processed by this matchmaker; FOR checking scheduling results */
    private int totalNumOfNondelayedLocalTasks = 0;

    /** Total number of delayed Tasks processed by this matchmaker; FOR checking scheduling results */
    private int totalNumOfDelayedLocalTasks = 0;

    // --- Partner Task category ---
    /** Task is considered as successful if Task.getStatus() == BGQoSMessage.SUCCESS */
    private int numOfSuccessProcessedPartnerTask = 0;

    /** Task is considered as failed if:
     * (1) no resource to send to, or (2) Task.getStatus() != BGQoSMessage.SUCCESS
     */
    private int numOfFailedProcessedPartnerTask = 0;

    /** Total number of nondelayed Tasks processed by this matchmaker; FOR checking scheduling results */
    private int totalNumOfNondelayedPartnerTasks = 0;

    /** Total number of delayed Tasks processed by this matchmaker; FOR checking scheduling results */
    private int totalNumOfDelayedPartnerTasks = 0;

    private double avgQueuingTime = 0;

    private static Log log = LogFactory.getLog(MatchMaker.class);

    private ConcurrentHashMap<String, String> receivedTaskIdMap;
    private ConcurrentHashMap<String, String> undeliveredTaskIdMap;



/**
     * Creates a new instance of MatchMaker
     */
    public MatchMaker(BGQoSEntity BGQoS, String localPolicy) throws Exception {

    this.BGQoS         = BGQoS;
    this.BGQoSIdentity   = BGQoS.getBGQoSIdentity();
    this.scheduleList    = new LinkedList();
    this.matchMakerPolicy = localPolicy;
```

```
this.receivedTaskIdMap = new ConcurrentHashMap<String, String>();
this.undeliveredTaskIdMap = new ConcurrentHashMap<String, String>();
}

/**
 * Once end of batch of Task processing, issued from TaskSubmitter, through ModuleController * (not necessarily end of simulation iteration)
 */

public void caculateStatistic (double newArrivalTardiness) {

this.totalTaskTardiness = newArrivalTardiness;

double TaskUsage = BGQoSToolkit.convertAtomicLongToDouble(this.BGQoS.getStorage().getTaskUsage());
    double resUsage = this.BGQoS.getStorage().getTotalNumOfPEs().get() * BGQoSMediator.getSystemTime();

    this.BGQoS.getStorage().setResUsage(BGQoSToolkit.convertDoubleToAtomicLong(resUsage));

    double resource Utilization = TaskUsage / resUsage;

this.BGQoS.getStorage().setResourceUtilization(BGQoSToolkit.convertDoubleToAtomicLong(resourceUtilization));


MMResult mmResult = new MMResult();

mmResult.setBGQoSIdentity(this.BGQoSIdentity);
mmResult.setMatchMakerPolicy(this.matchMakerPolicy);
mmResult.setTotalTaskTardiness(this.totalTaskTardiness);
mmResult.setTotalTaskWeight(this.totalTaskWeight);
mmResult.setTotalSchedulingTime(this.totalSchedulingTime);
mmResult.setResourceUptime(BGQoSMediator.getSystemTime());
mmResult.setTotalTaskResponseTime(this.totalTaskResponseTime);
mmResult.setTotalTaskWaitingTime(this.totalTaskWaitingTime);
mmResult.setTotalWeightedResponseTime(this.totalWeightedResponseTime);
mmResult.setTotalTaskCPUTime(this.totalTaskCPUTime);
mmResult.setTotalWeightedTaskCPUTTime(this.totalWeightedTaskCPUTTime);
mmResult.setTotalSlowdown(this.totalSlowdown);
mmResult.setTotalWeightedSlowdown(this.totalWeightedSlowdown);

    // local Task category
mmResult.setNumOfFailedProcessedLocalTask(this.numOfFailedProcessedLocalTask);
mmResult.setNumOfReceivedLocalTasks(this.numOfReceivedLocalTasks);
mmResult.setNumOfSuccessProcessedLocalTask(this.numOfSuccessProcessedLocalTask);
mmResult.setTotalNumOfDelayedLocalTasks(this.totalNumOfDelayedLocalTasks);
mmResult.setTotalNumOfNondelayedLocalTasks(this.totalNumOfNondelayedLocalTasks);
```

```
      // Partner Task category
mmResult.setNumOfFailedProcessedPartnerTask(this.numOfFailedProcessedPartnerTask);
mmResult.setNumOfReceivedPartnerTasks(this.numOfReceivedPartnerTasks);
mmResult.setNumOfSuccessProcessedPartnerTask(this.numOfSuccessProcessedPartnerTask);
mmResult.setTotalNumOfDelayedPartnerTasks(this.totalNumOfDelayedPartnerTasks);
mmResult.setTotalNumOfNondelayedPartnerTasks(this.totalNumOfNondelayedPartnerTasks);

      this.BGQoS.getStorage().setMMResult(mmResult);
   }


   /****************************************
    access method for external invoking
   ****************************************/

   /**
    * Info event from ModuleController: Task already sent to specific resource, which is made by
MatchMaker
    */
   public void TaskAlreadySentToLocalResource() {

   this.numOfTaskWaitingForSchedule--;
   this.numOfExistingSchedules--;

      // do another scheduling round
      if(numOfExistingSchedules == 0){
         this.callSchedule();
      }
   }

   /**
    * Info event from ModuleController: local Task can't be sent to specific resource (resource invalid or Id
available), which is made by MatchMaker
    */
   public void localTaskNotSentToResource() {
                this.numOfFailedProcessedLocalTask += 1;
   }

   /**
    * Info event from ModuleController: Partner Task can't be sent to specific resource (resource invalid or
Id unavailable), which is made by MatchMaker
    */
   public void PartnerTaskNotSentToResource() {
                this.numOfFailedProcessedPartnerTask += 1;
   }

   /**
```

```
 * Task is neither submitted to local resource nor to global resources
 */
public void TaskUndelivered (TaskInfo gi) {

        if (this.undeliveredTaskIdMap.containsKey(gi.getGlobalTaskID())) {

                return;

        } else {

                If (gi.getOriginalBGQoSId().equals(this.BGQoS.getBGQoSIdentity())) {
                        this.numOfReceivedLocalTasks++;
                } else {
                        this.numOfReceivedPartnerTasks++;
                }
        }

}

/**
 * Task submitted from ModuleController to get scheduled
 * It's also the place to calculate resource "execution load and max load"
 */
public void TaskScheduled(TaskInfo gi) {

        if (gi.getOriginalBGQoSId().equals(this.BGQoS.getBGQoSIdentity())) {

                // Send local Task's GlobalID to the MatchMaker's TaskQueue
                this.localTaskSubmitted(gi.getGlobalTaskID());

        } else {

                // Send Partner Task's GlobalID to the MatchMaker's TaskQueue
                this.PartnerTaskSubmitted(gi.getGlobalTaskID());
        }

}


private void localTaskSubmitted(String item) {

this.numOfReceivedLocalTasks++;
this.numOfTaskWaitingForSchedule++;

    // add Task to queue
    // Put the Task into Task queue, the Tasks in a queue would be invoked by MatchMaker's scheduling
policy
```

```
this.insertTaskToProcessingQueue(item);

   // making schedule based on specific algorithms for queued Tasks

this.callSchedule();

   }

   private void PartnerTaskSubmitted(String item) {

                this.numOfReceivedPartnerTasks++;
     this.numOfTaskWaitingForSchedule++;

     // add Task to queue
     // Put the Task into Task queue, the Tasks in queue would be invoked by MatchMaker's scheduling
policy

this.insertTaskToProcessingQueue(item);

     // making schedule based on specific algorithms for queued Tasks

this.callSchedule();
   }



   /**
    * Insert a newly submitted Task to the MatchMaker TaskQueue,
    */

   private void insertTaskToProcessingQueue(String item) {

                if(this.localTaskQueue.contains(item)) {
                             return;
                }

                // determine how the Tasks are appended to queue according to adopted MatchMaker policy
                if(this.matchMakerPolicy.equals(BGQoSMessage.PolicyFCFS)) {
                             // FCFS
                             this.addTaskToLocalQueue(item);

                } else if(this.matchMakerPolicy.equals(BGQoSMessage.PolicyQOS)) {
                             // QOS, TaskQueue sorting appended


                             if(this.localTaskQueue.size() == 0) {

                                          this.addTaskToLocalQueue(item);
```

```
                        } else {

                                // fetch newTask's estimated execution time (EST)
                                int index = -1;
                                TaskInfo newTaskInfo = TaskCenterManager.getTaskInfobyTaskId(item);
                                double newEST = newTaskInfo.getEstimatedComputationTime();

                                for(int i = 0; i < this.localTaskQueue.size(); i++) {

                                        TaskInfo currentTaskInfo = TaskCenterManager.getTaskInfobyTaskId(this.localTaskQueue.get(i));
                                        double currentEST = currentTaskInfo.getEstimatedComputationTime();
                                        if(newEST <= currentEST){

                                                index = i;
                                                break;
                                        }
                                } // end of comparison loop

                                if(index == -1) {
                                        this.addTaskToLocalQueue(item);
                                } else {
                                        this.addTaskToLocalQueue(index, item);
                                }
                        }

                } else if(this.matchMakerPolicy.equals(BGQoSMessage.PolicyEasyBF)) {
                        // FCFS-like queue for EASY Backfilling
                        this.addTaskToLocalQueue(item);

                } else {
                        // FCFS queue is the default policy
                        this.addTaskToLocalQueue(item);

                }

        }

        private void addTaskToLocalQueue(String TaskId) {
                this.localTaskQueue.add(TaskId);
        }

        private void addTaskToLocalQueue(int index, String TaskId) {
                this.localTaskQueue.add(index, TaskId);
```

```
    }


    private boolean removeTaskFromLocalQueue(String TaskId) {

            GlobalStorage.test_counter_3.incrementAndGet();

TaskInfo TaskInfo = TaskCenterManager.getTaskInfobyTaskId(TaskId);
TaskCenterManager.TaskProcessing(TaskInfo.getGlobalTaskID(),
this.BGQoSIdentity, TaskInfo);


            return this.localTaskQueue.remove(TaskId);
    }

    /**
     * Estimated processing time for already queued Tasks on this node
     */
    public double estimatedTimeToExecuteLocalTaskQueue() {

            double estimatedTime = 0;
            double load = 0;

            for(String TaskId : localTaskQueue) {
                    TaskInfo TaskInfo = TaskCenterManager.getTaskInfobyTaskId(TaskId);
                    load += TaskInfo.getComputationalLength() * TaskInfo.getNumPE();
            }

            estimatedTime = load / this.BGQoS.getStorage().getTotalNumOfPEs().get();

            return estimatedTime;
    }

    /**
     * Estimated processing time for this node's queued    *
     */
    public double estimatedTimeToExecuteLocalTaskQueue(String targetTaskId) {

            double estimatedTime = 0;
            double load = 0;

            int index = localTaskQueue.indexOf(targetTaskId);

            for(int i = 0; i < index; i++) {
                    TaskInfo                                TaskInfo                                =
TaskCenterManager.getTaskInfobyTaskId(localTaskQueue.get(i));
                    load += TaskInfo.getComputationalLength() * TaskInfo.getNumPE();
```

```
                }

                estimatedTime = load / this.BGQoS.getStorage().getTotalNumOfPEs().get();

                return estimatedTime;
        }


        /**
         * Estimated processing time for already "promised" Tasks
         */
        public double estimatedTimeToExecuteShadowTaskQueue() {

                double estimatedTime = 0;
                double load = 0;

                Iterator <Map.Entry<String, Double>> iter = shadowTaskQueue.entrySet().iterator();

                while (iter.hasNext()) {
                        Map.Entry<String, Double> entry = iter.next();
                        String TaskId = entry.getKey().trim();

                        TaskInfo TaskInfo = TaskCenterManager.getTaskInfobyTaskId(TaskId);
                        double acceptanceApproveProbability = entry.getValue().doubleValue();

                        load += TaskInfo.getComputationalLength() * TaskInfo.getNumPE() *
                                acceptanceApproveProbability                                    *
BGQoSParam.weightOfShadowTaskQueue;
                }

                estimatedTime = load / this.BGQoS.getStorage().getTotalNumOfPEs().get();

                return estimatedTime;
        }

        /**
         * reserve an acceptance decision
         *
         * @param TaskId
         * @param acceptanceApproveProbability
         */
        public void appendAcceptanceDecision(String TaskId, double acceptanceApproveProbability) {
                this.shadowTaskQueue.put(TaskId, new Double(acceptanceApproveProbability));
        }

        /**
         * revoke an acceptance decision
         *
```

```
     * @param TaskId
     */

public void revokeAcceptanceDecision(String TaskId) {
            this.shadowTaskQueue.remove(TaskId);
   }

   /**
    * Update the average queuing time of this node
    */
   public void updateQueuingTime() {

            double queuingTime = 0;
            double numOfTasks  = 0;
            double systemTime  = BGQoSMediator.getSystemTime();
            int sizeOfLocalTaskQueue = localTaskQueue.size();

            if(sizeOfLocalTaskQueue > 0) {

                     for(String TaskId : localTaskQueue) {

                              TaskInfo TaskInfo = TaskCenterManager.getTaskInfobyTaskId (TaskId);

                              double queuingStartTime = TaskInfo.getQueuingStartTime();
                              queuingTime += systemTime - queuingStartTime;

                              TaskInfo.setQueuingTime(systemTime - queuingStartTime);
                     }
                     numOfTasks += localTaskQueue.size();

            } else {

                     Vector<TaskInfo>                    exedTaskInfo                    =
TaskCenterManager.getTask_processingNode(this.BGQoSIdentity, TaskCenterManager.EXECUTED);
                     for(TaskInfo TaskInfo : exedTaskInfo) {
                              queuingTime += TaskInfo.getQueuingTime();
                     }
                     numOfTasks += exedTaskInfo.size();

            }

            if(numOfTasks != 0) {
                     this.avgQueuingTime = queuingTime / numOfTasks;
            } else {
                     this.avgQueuingTime = 0;
            }

   }
```

```
/**
 * Fetch one Task, which may wait long time before getting scheduled and executed, for rescheduling processing
 * @return
 */
public Vector<String> searchTasksForRescheduling() {

int sizeOfLocalTaskQueue = localTaskQueue.size();

if(sizeOfLocalTaskQueue < 1) {
        return null;
}

double systemTime  = BGQoSMediator.getSystemTime();
Vector<String> toRescheduleTasks = new Vector<String>();

try {

        // update the this resources instant average Task queuing time
        this.updateQueuingTime();

        for(String TaskId : localTaskQueue) {

                TaskInfo TaskInfo = TaskCenterManager.getTaskInfobyTaskId(TaskId);

                // if Task's requirement beyonds resource profile
                if(!check_TaskMatchResource(TaskInfo)) {
                        toRescheduleTasks.add(TaskId);
                        continue;
                }

                double queuingStartTime = TaskInfo.getQueuingStartTime();
                double queuingTime = systemTime - queuingStartTime;
                TaskInfo.setQueuingTime(queuingTime);


                if(this.avgQueuingTime != 0) {
                        double relativeQueuingDelay = queuingTime / this.avgQueuingTime;

                        if(relativeQueuingDelay >= BGQoSParam.systemReschedulingCoefficient) {
                                toRescheduleTasks.add(TaskId);
                        }

                } else {
                        // if hosting node's avg queuing time is zero or unavailable,
```

```
                                        // then there is no need to reschedule Tasks
                                }

                        }

                        For (int i = 0; i < toRescheduleTasks.size(); i++)  {

                                for (int j = i + 1; j < toRescheduleTasks.size(); j ++)  {

                                        if
(TaskCenterManager.getTaskInfobyTaskId(toRescheduleTasks.get(i)).getQueuingTime() <

   TaskCenterManager.getTaskInfobyTaskId(toRescheduleTasks.get(j)).getQueuingTime())  {

                                                String tmpId = toRescheduleTasks.get(i);
                                                toRescheduleTasks.set(i, toRescheduleTasks.get(j));
                                                toRescheduleTasks.set(j, tmpId);
                                        }
                                }
                        }

   } catch (Exception e) {
                        e.printStackTrace();
                        System.exit(0);
   }

   return toRescheduleTasks;

   }


   /**
    * Remove a TaskItem from the MatchMaker and re-schedule it to a remote node
    * @param TaskId
    * @return
    */
   public synchronized boolean TaskReschedule(String TaskId) {

                        if(this.localTaskQueue.contains(TaskId)) {

                                if(TaskId.equals(this.BGQoS.getBGQoSIdentity())) {
                                        this.numOfReceivedLocalTasks--;

                                } else {
                                        this.numOfReceivedPartnerTasks--;
                                }

//                                      this.receivedTaskIdMap.remove(TaskId);
```

```
                        this.numOfTaskWaitingForSchedule--;

                        return removeTaskFromLocalQueue(TaskId);


                } else {
                        return false;
                }
        }

        /**
         * Info event from ModuleController: Task already finished by resource
         */
        public void TaskFinishedConfirmation(TaskInfo gi) {

                if(gi.getOriginalBGQoSId().equals(this.BGQoS.getBGQoSIdentity())) {
                        this.localTaskFinishedConfirmation(gi);
                } else {
                        this.PartnerTaskFinishedConfirmation(gi);

        GlobalStorage.findBGQoSById(gi.getOriginalBGQoSId()).getPartnerMonitor().TaskDelegationCompleteBy
RemoteNode(gi);
                }
        }

        /**
         * Local Task executed
         * @param TaskInfo
         */
        private void localTaskFinishedConfirmation(TaskInfo gi) {

        if(gi.getTaskStatus() == BGQoSMessage.Task_SUCCESS) {
                        this.numOfSuccessProcessedLocalTask += 1;
                } else {
                        this.numOfFailedProcessedLocalTask += 1;
                }

                // single Task tardiness
            double TaskTardiness = gi.getTardiness();

            // Task response time
            // NOTICE: Task is released at "gi.getTaskStartTime()", doesn't mean the execution will start, it could
be delayed
            double TaskResponse = gi.getTask().getFinishTime() - gi.getArrivalTime();

            // calculate & update total weighted and normal slow down
            // NOTICE: Task getAcutalCPUTime reflect how much time used by a Task (each required PE runs the
same time)
```

```
        double TaskWeight = gi.getNumPE() * gi.getTask().getActualCPUTime();
        double TaskSlowdown = TaskResponse / gi.getTask().getActualCPUTime();

        this.totalTaskWeight  += TaskWeight;

        this.totalTaskResponseTime     += TaskResponse;
        this.totalTaskWaitingTime += TaskResponse - gi.getTask().getActualCPUTime();

        this.totalWeightedResponseTime += TaskWeight * TaskResponse;
        this.totalTaskCPUTime          += gi.getTask().getActualCPUTime();
        this.totalWeightedTaskCPUTTime  += TaskWeight * gi.getTask().getActualCPUTime();

        if(Double.isInfinite(TaskSlowdown)) {
                // to handling unexpected errors, e.g., no Task actualCPUTime available, replace it by the mean
(averaged) of Task slowdown
                // for example, after 500 Task executed, if current total slowdown is 1000,
                // then the mean Task slowdown is 2, thus the totalslowdown is add-up by 2 (another mean
slowdown)
                this.totalSlowdown                      +=          this.totalSlowdown          /
(this.numOfSuccessProcessedLocalTask + this.numOfSuccessProcessedPartnerTask);
                this.totalWeightedSlowdown                      +=      this.totalWeightedSlowdown      /
(this.numOfSuccessProcessedLocalTask + this.numOfSuccessProcessedPartnerTask);
        } else {
                this.totalSlowdown                      += TaskSlowdown;
                this.totalWeightedSlowdown    += TaskWeight * TaskSlowdown;
        }

        this.BGQoS.getStorage().updateUsage(TaskWeight);

        // update corresponding resource profile from the persist storage
        LinkedList<ResourceInfo> localResourceInfoList = this.BGQoS.getStorage().getResourceInfoList();

        for (ResourceInfo ri : localResourceInfoList){

          if (gi.getTask().getResourceID() == ri.getResource().getResourceID()){
            // lower the load of resource, update info about overall resource tardiness and exit cycle
            ri.lowerResInExec(gi);
            ri.setTotalTardinessOfFinishedTasks(ri.getTotalTardinessOfFinishedTasks() + TaskTardiness);

            if(TaskTardiness <= 0.0){

ri.setNumOfPreviousFinishedNondelayedTasks(ri.getNumOfPreviousFinishedNondelayedTasks() + 1);
              totalNumOfNondelayedLocalTasks++;

            }else{
              totalNumOfDelayedLocalTasks++;
            }
            break;
```

```
        }
      }

      // some resource is probably available - try send next Task according to schedule
      if(numOfExistingSchedules == 0){
        this.callSchedule();
      }
    }

    /**
     * Partner Task executed
     * @param TaskInfo
     */
    private void PartnerTaskFinishedConfirmation(TaskInfo gi) {

    if(gi.getTaskStatus() == BGQoSMessage.Task_SUCCESS) {
                        this.numOfSuccessProcessedPartnerTask += 1;
                } else {
                        this.numOfFailedProcessedPartnerTask += 1;
                }

                // single Task tardiness
      double TaskTardiness = gi.getTardiness();

      // Task response time

      double TaskResponse = gi.getTask().getFinishTime() - gi.getArrivalTime();

      // calculate & update total weighted and normal slow down
      // NOTICE: Task getAcutalCPUTime reflect how much time is used by a Task (each required PE runs the
same time)

      double TaskWeight = gi.getNumPE() * gi.getTask().getActualCPUTime();
      double TaskSlowdown = TaskResponse / gi.getTask().getActualCPUTime();

      this.totalTaskWeight                    += TaskWeight;

      this.totalTaskResponseTime     += TaskResponse;
      this.totalTaskWaitingTime += TaskResponse - gi.getTask().getActualCPUTime();

      this.totalWeightedResponseTime += TaskWeight * TaskResponse;
      this.totalTaskCPUTime          += gi.getTask().getActualCPUTime();
      this.totalWeightedTaskCPUTTime  += TaskWeight * gi.getTask().getActualCPUTime();

      if(Double.isInfinite(TaskSlowdown)) {
                // to handling unexpected errors, e.g., no Task actualCPUTime available, replace it by the mean
(averaged) of Task slowdown
                // for example, after 500 Task executed, if current total slowdown is 1000,
```

```
                // then the mean Task slowdown is 2, thus the totalslowdown is add-up by 2 (another mean
slowdown)
                this.totalSlowdown                    +=          this.totalSlowdown          /
(this.numOfSuccessProcessedLocalTask + this.numOfSuccessProcessedPartnerTask);
                this.totalWeightedSlowdown                    +=    this.totalWeightedSlowdown    /
(this.numOfSuccessProcessedLocalTask + this.numOfSuccessProcessedPartnerTask);
    } else {
                this.totalSlowdown                    += TaskSlowdown;
                this.totalWeightedSlowdown    += TaskWeight * TaskSlowdown;
    }

    this.BGQoS.getStorage().updateUsage(TaskWeight);

    // update corresponding resource profile from the persist storage
    LinkedList<ResourceInfo> localResourceInfoList = this.BGQoS.getStorage().getResourceInfoList();

    for (ResourceInfo ri : localResourceInfoList){

      if (gi.getTask().getResourceID() == ri.getResource().getResourceID()){
        // lower the load of resource, update info about overall resource tardiness and exit cycle
        ri.lowerResInExec(gi);
        ri.setTotalTardinessOfFinishedTasks(ri.getTotalTardinessOfFinishedTasks() + TaskTardiness);

        if(TaskTardiness <= 0.0){

ri.setNumOfPreviousFinishedNondelayedTasks(ri.getNumOfPreviousFinishedNondelayedTasks() + 1);
          totalNumOfNondelayedPartnerTasks++;

        }else{
          totalNumOfDelayedPartnerTasks++;
        }
        break;
      }
    }

    if(numOfExistingSchedules == 0){
                this.callSchedule();
    }
  }

  /**
  * When new response made, Task owner (ModuleController) will be notified
  */
  private void sendResponseToTaskOwner(TaskInfo item) {

  this.BGQoS.getModuleController().updateFromMatchMaker(item);
  }
```

```
  /**
   * Starts scheduling according to prepared schedule/queue
   */
  private boolean scheduleTasks(){

    // pick up the corresponding local scheduling policy
    if(matchMakerPolicy.equals(BGQoSMessage.PolicyExistingSchedule)){
this.numOfExistingSchedules = useSchedule();
      return true;

    } else if(matchMakerPolicy.equals(BGQoSMessage.PolicyFCFS)){
            this.numOfExistingSchedules = useFCFS();
            return true;

    } else if(matchMakerPolicy.equals(BGQoSMessage.PolicyQOS)){
            this.numOfExistingSchedules = useQOS();
      return true;

    } else if(matchMakerPolicy.equals(BGQoSMessage.PolicyEasyBF)){
      numOfExistingSchedules = useEASY();
      return true;

    } else {
            this.numOfExistingSchedules = useFCFS();
            return true;
    }

  }


  /**
   * Call the existing schedule policy and record the used time
   */
  private void callSchedule() {

            Date d = new Date();
    clockBeforeMakingSchedule = d.getTime();

    //  make next round schedules
    scheduleTasks();

    Date d2 = new Date();
    clockAfterMakingSchedule = d2.getTime();

    totalSchedulingTime += clockAfterMakingSchedule - clockBeforeMakingSchedule;
  }
```

```
/************************
 * Scheduling Policies
 ************************/

/**
 * FCFS algorithm managing incoming Task queue
 */
private int useFCFS(){

   int successSched = 0;
   ResourceInfo selectedResourceInfo = null;

   while(!localTaskQueue.isEmpty()){

      LinkedList<ResourceInfo> localResourceInfoList = this.BGQoS.getStorage().getResourceInfoList();

      // Refresh (update to latest numOfFreePE) the numOfFreVirtualPE for anticipating scheduling
process
      for (ResourceInfo ri : localResourceInfoList){
            ri.setNumOfVirtualFreePE(new AtomicInteger(ri.getNumOfFreePE()));
      }

      // Retrieve the Task from TaskQueue
      TaskInfo TaskInfo = TaskCenterManager.getTaskInfobyTaskId(this.selectTaskFromQueue());

      // If Task requirement exceeds resource capability, this Task cannot be scheduled this time it is set to
status FAILED directly
      if((TaskInfo == null) || (!this.check_TaskMatchResource(TaskInfo))) {


            TaskInfo.setTargetResourceID(-1);
            removeTaskFromLocalQueue(TaskInfo.getGlobalTaskID());
            this.sendResponseToTaskOwner(TaskInfo);

        TaskInfo = null;
        continue;
      }

      //  IMPORTANT: selected resourceInfo MUST be reset for each to-process TaskInfo
      selectedResourceInfo = null;

      //  FCFS: select a resource (the first candidate), which match Task's PE requirement and has the best
MIPS
      for (ResourceInfo ri : localResourceInfoList){

        if((ri.getNumOfVirtualFreePE().get() >= TaskInfo.getNumPE()) &&
                  (ri.getNumOfTotalPE() >= TaskInfo.getNumPE())) {
```

```
                selectedResourceInfo = ri;
                ri.setNumOfVirtualFreePE(new AtomicInteger(ri.getNumOfVirtualFreePE().get() -
TaskInfo.getNumPE()));

                // resource "First Fit" selection
                break;
            }
        }

        if(selectedResourceInfo != null){

                // Current Task marked to be sent to selected resource
            TaskInfo.setTargetResourceID(selectedResourceInfo.getResource().getResourceID());

                // Current Task removed from queue
            this.removeTaskFromLocalQueue(TaskInfo.getGlobalTaskID());

            // Important: resource profile notified with new Task
            selectedResourceInfo.addTaskInfoInExec(TaskInfo);

            // ModulerController notified with new MatchMaker decision
            this.sendResponseToTaskOwner(TaskInfo);
            successSched += 1;

        } else {


                break;

        }

        TaskInfo = null;

    } // exit loop TaskQueue

    return successSched;
}

/**
 * QOS algorithm managing incoming Task queue
 */
private int useQOS(){

return this.useFCFS();
}
```

```
/**
 * EasyBackfilling algorithm managing incoming Task queue
 */
private int useEASY(){

int successSched = 0;
  ResourceInfo selectedResourceInfo = null;
  boolean backfillingNeeded = false;

  while(!localTaskQueue.isEmpty()){

    LinkedList<ResourceInfo> localResourceInfoList = this.BGQoS.getStorage().getResourceInfoList();

    // Refresh (update to latest numOfFreePE) the numOfFreVirtualPE for anticipating scheduling process
    for (ResourceInfo ri : localResourceInfoList){
        ri.setNumOfVirtualFreePE(new AtomicInteger(ri.getNumOfFreePE()));
    }

    // Retrieve the Task from TaskQueue
    TaskInfo TaskInfo = TaskCenterManager.getTaskInfobyTaskId(this.selectTaskFromQueue());

    // If Task requirement exceeds resource capability, this Task cannot be scheduled this time it is set to status FAILED directly
    if((TaskInfo == null) || (!this.check_TaskMatchResource(TaskInfo))) {

        TaskInfo.setTargetResourceID(-1);

        this.removeTaskFromLocalQueue(TaskInfo.getGlobalTaskID());

      this.sendResponseToTaskOwner(TaskInfo);

      TaskInfo = null;
      continue;
    }

    // Selected resourceInfo MUST be reset for each to-process TaskInfo
    selectedResourceInfo = null;

    // Select a resource (the top ranked candidate), which match Task's PE requirement and has the best MIPS
    for (ResourceInfo ri : localResourceInfoList){

      if((ri.getNumOfVirtualFreePE().get() >= TaskInfo.getNumPE()) &&
              (ri.getNumOfTotalPE() >= TaskInfo.getNumPE())) {

        selectedResourceInfo = ri;
        ri.setNumOfVirtualFreePE(new AtomicInteger(ri.getNumOfVirtualFreePE().get() -
```

```
TaskInfo.getNumPE()));

                    break;
                }
            }

            if(selectedResourceInfo != null){

                    // Current Task marked to be sent to selected resource
                TaskInfo.setTargetResourceID(selectedResourceInfo.getResource().getResourceID());

                    // Current Task removed from queue
                this.removeTaskFromLocalQueue(TaskInfo.getGlobalTaskID());

                // Important: resource profile notified with new Task
                selectedResourceInfo.addTaskInfoInExec(TaskInfo);

                this.sendResponseToTaskOwner(TaskInfo);
                successSched += 1;

            } else {

                    // Here with the first element of the queue could be executed successfully in local resource
                    // but the corresponding resource is not ready yet
                    // therefore, the TaskQueue checking will be blocked here, no matter whether another Task
inside the queue
                    // could be executed now, it won't invoked in FCFS
                    backfillingNeeded = true;
                    break;

            }

            TaskInfo = null;

        } // exit loop TaskQueue

        // starting backfilling phase
        if(backfillingNeeded && this.localTaskQueue.size() > 1) {

                String headTaskId    = this.localTaskQueue.get(0);
                TaskInfo headTaskInfo = TaskCenterManager.getTaskInfobyTaskId(headTaskId);
                ResourceInfo reservedResourceInfo = this.findReservedResource(headTaskInfo);

                // looping all other Tasks (except the first one) of MatchMaker's TaskQueue
                for(int j = 1; j < this.localTaskQueue.size(); j++) {
                        String currentTaskId    = this.localTaskQueue.get(j);
                        TaskInfo                                currentTaskInfo                                =
```

```
TaskCenterManager.getTaskInfobyTaskId(currentTaskId);

                        // jump over Tasks which will never be executed because of asking more PEs than
resource's capability
                        if(currentTaskInfo.getNumPE() >= reservedResourceInfo.getNumOfTotalPE()) {
                                continue;
                        }

                        ResourceInfo resInfo = this.findResourceBF(currentTaskInfo, headTaskInfo,
reservedResourceInfo);

                        if(resInfo != null){

                                // Current Task marked to be sent to selected resource
            currentTaskInfo.setTargetResourceID(resInfo.getResource().getResourceID());

              // Current Task removed from queue
            this.removeTaskFromLocalQueue(currentTaskInfo.getGlobalTaskID());

            // Important: resource profile notified with new Task
            resInfo.addTaskInfoInExec(currentTaskInfo);

            // ModulerController notified with new MatchMaker decision
            this.sendResponseToTaskOwner(currentTaskInfo);

            backfillingNeeded = false;
            successSched += 1;
            // Important: one Task has been backfilled, therefore MatchMaker's TaskQueue size is decreased
            j--;

          }
              }
    }

    return successSched;
  }


  /**
   * Auxiliary method for EASY Backfilling
   */
  @SuppressWarnings("unchecked")
  private ResourceInfo findResourceBF(TaskInfo newTask, TaskInfo blockedFirstTask, ResourceInfo
reservedResForBlockedFirstTask){

    ResourceInfo r_cand = null;
    int r_cand_speed = 0;
```

```
LinkedList localResourceInfoList = this.BGQoS.getStorage().getResourceInfoList();

for (int j=0; j < localResourceInfoList.size(); j++) {

  ResourceInfo ri = (ResourceInfo) localResourceInfoList.get(j);
  if(ri.getNumOfFreePE() < 1) {
        continue;
  }


  if(ri.getNumOfFreePE() >= newTask.getNumPE() && ri.getResource().getResourceID() != reservedResForBlockedFirstTask.getResource().getResourceID()){

        int speed = ri.getResource().getMIPSRatingOfOnePE();
    if(speed >= r_cand_speed){
      r_cand = ri;
      r_cand_speed = speed;
    }

  } else if (ri.getNumOfFreePE() >= newTask.getNumPE() && ri.getResource().getResourceID() == reservedResForBlockedFirstTask.getResource().getResourceID()){

        // precondition:
        // shadow time: when enough nodes will be available for the first queued(currently blocked) Task
        // extra PE: if the first Task does not need all available PEs, the ones left over are the extra PEs
        double newTaskEstimatedFinishTime = BGQoSMediator.getSystemTime() + (newTask.getComputationalLength()/ri.getResource().getMIPSRatingOfOnePE());
        double shadowTime = ri.getEarliestStartTime();
        int extraPE = ri.getNumOfTotalPE() - blockedFirstTask.getNumPE();
        double minPE = Math.min(ri.getNumOfFreePE(), extraPE);

//
        // to determine whether a being checked Task can be fit backfilling, need to check as follows:
        // Either, it requires no more than currently free PEs on this resource, and will terminate by the shadow time
        // Or, it requires no more than minimum of currently free PEs and extra PEs, namely it requires no more than min(freePEs_onResource, extra_PE)

        if(newTaskEstimatedFinishTime <= shadowTime){
//              log.info("******************\n******************");
      int speed = ri.getResource().getMIPSRatingOfOnePE();
      if(speed > r_cand_speed){
        r_cand = ri;
        r_cand_speed = speed;
      }

    } else if(newTask.getNumPE() <= minPE) {
```

```
//                log.info("******************\n******************");
                  int speed = ri.getResource().getMIPSRatingOfOnePE();
               if(speed > r_cand_speed){
                  r_cand = ri;
                  r_cand_speed = speed;
               }
            }
         }
      }

      // save the ResourceInfo List information back
      this.BGQoS.getStorage().setResourceInfoList(localResourceInfoList);

      return r_cand;
   }


   /**
    * Find the reserved resource for the first Task of the queue, which is blocked therefore need backfilling (if
multi-resources on one node)
    *
    * Auxiliary method for EASY Backfilling
    */
   @SuppressWarnings("unchecked")
   private ResourceInfo findReservedResource(TaskInfo grsv){

      double est = Double.MAX_VALUE;
      ResourceInfo found = null;

      LinkedList localResourceInfoList = this.BGQoS.getStorage().getResourceInfoList();

      for (int j=0; j < localResourceInfoList.size(); j++){

         ResourceInfo ri = (ResourceInfo) localResourceInfoList.get(j);

         if(ri.getNumOfTotalPE() >= grsv.getNumPE()){

                  // find the resource with earliest start time
            double ri_est = ri.getEarliestStartTimeForTaskInfo(grsv, BGQoSMediator.getSystemTime());
            // select minimal EST
            if(ri_est <= est){
               est = ri_est;
               found = ri;
            }

         } else {
                  continue; // this is not suitable machine
         }
```

```
    }

    // save the ResourceInfo List information back
    this.BGQoS.getStorage().setResourceInfoList(localResourceInfoList);

    return found;
}

/**
 * Select Task from the TaskQueue of MatchMaker
 * @return
 */
private String selectTaskFromQueue() {

if(this.localTaskQueue.size() > 0) {
        return this.localTaskQueue.get(0);
} else {
        return null;
}

}

/**
 * Check whether TaskInfo can be satisfied by local resources (TaskInfo)
 * by receiving queries from MatchMaker itself
 */
private boolean check_TaskMatchResource(TaskInfo TaskInfo) {

if((!this.BGQoS.getBGQoSMemory().equals(TaskInfo.getTask().getMemoryRequired())) ||
                (TaskInfo.getNumPE() > this.BGQoS.getStorage().getTotalNumOfPEs().get())) {
        return false;

} else {
        return true;
}

}

/**
 * Check whether TaskInfo can be satisfied by local resources
 * by receiving queries from external scheduling components, such as  Controller
 *
 * @param TaskReliability
 * @param TaskNumPE
 * @return
 */
public boolean check_TaskMatchResource(String TaskReliability , int TaskNumPE) {
```

```
if((!this.BGQoS.getBGQoSReliability().equals(TaskReliability )) ||
                        (TaskNumPE > this.BGQoS.getStorage().getTotalNumOfPEs().get())) {
            return false;

} else {
            return true;
}

}

public boolean check_TaskInstantMatchResource(String TaskReliability , int TaskNumPE) {

int numFreePE = this.BGQoS.getStorage().getTotalVirtualFreePEs();

if((!this.BGQoS.getBGQoSOS().equals(TaskReliability )) || (TaskNumPE > numFreePE)) {
            return false;

} else {
            return true;
}

}

/**
 * Check whether still available resources(PEs) for incoming Task request
 *
 * @param TaskInfo
 * @return true IF matches!
 */
private boolean check_resourceAvailable(TaskInfo TaskInfo) {

int          freePE          =          this.BGQoS.getStorage().getTotalNumOfPEs().get()          -
this.BGQoS.getStorage().getTotalActivePEs().get();

if(freePE > TaskInfo.getNumPE()) {
            return true;
} else {
            return false;
}
}


/**
 * Check whether still available resources(PEs) for incoming Task request
 * by receiving queries from external components, such as  Controller
 *
 * @param TaskNumPE
 * @return
```

```
*/
public boolean check_resourceAvailable(int TaskNumPE) {

int freePE = this.BGQoS.getStorage().getTotalNumOfPEs().get() -
this.BGQoS.getStorage().getTotalActivePEs().get();

if(freePE > TaskNumPE) {
        return true;
} else {
        return false;
}
}



/**
 * This method updates Task priority P_j according to Flexible Backfilling strategy.
 * @param queue Incoming queue of Tasks
 * @param time Current simulation time
 * @deprecated
 */
private void updateTaskPriority(LinkedList queue, double time){

  int bm = this.BGQoS.getStorage().getBestMachineMIPS().get();

  // sort the queue according to estimated exec. time
  Collections.sort(queue, new LengthComparator());

  // compute new priorities
  for(int i = 0; i < queue.size(); i++){
        TaskInfo gi = (TaskInfo) queue.get(i);
    // Aging
    double age_factor = 0.01;
    double p = 0.0;
    double age = time - gi.getArrivalTime();
    p += age_factor * age;

    // Deadline
    double deadline = gi.getDeadline();
    double estimated = gi.getEstimatedComputationTime();
    double nxtime = 0.0;
    double extime = 0.0;
    double k = 2.0; // reset
    double bme = gi.getEstimatedComputationalMIPS();
    double t = 0.0;
    nxtime = estimated * (bme/bm);
    extime = time + nxtime;
    t = deadline - k*nxtime;
```

```
        double max = 20.0;
        double min = 0.1;
        double a = (max - min)/(deadline - t);
        //double a = 1.0; // reset
        if(extime <= t) p+= min;
        if(t < extime && extime <= deadline) p += a * (extime - t) + min;
        if(extime > deadline) p += min;

        // Wait Minimization
        double boostvalue = 2.0; // reset
        // get the shortest Gridlet according to "estimated" parameter
        TaskInfo shortest = (TaskInfo) queue.getLast();
        double minext = shortest.getEstimatedComputationTime();
        p += (boostvalue * minext)/estimated;
        gi.setTaskPriority(p);
      }

}


/** Get number of already made schedulers by this MatchMaker */
public int getNumOfExistingSchedules() {
                return numOfExistingSchedules;
}

/**
 * Get total time used for making schedule generation, i.e. time =
 * Sum(clockAfterMakingSchedule - clockBeforeMakingSchedule)
 */
public double getTotalSchedulingTime() {
                return totalSchedulingTime;
}

/** Get total number of nondelayed Tasks processed by this matchmaker */
public int getTotalNumOfNondelayedLocalTasks() {
                return totalNumOfNondelayedLocalTasks;
}

/** Get total number of delayed Tasks processed by this matchmaker */
public int getTotalNumOfDelayedLocalTasks() {
                return totalNumOfDelayedLocalTasks;
}

/**
 * Get number of Tasks waiting for scheduling decision It will be decrease
 * only if TaskSubmiiter inform MatchMaker that the Task is already sent to
 * resource
 */
```

```
public int getNumOfTaskWaitingForSchedule() {
		return numOfTaskWaitingForSchedule;
}

/** Get total time used to execute all the Tasks */
public double getTotalTaskExecutionTime() {
		return totalTaskResponseTime;
}

public double getTotalTaskWaitingTime() {
		return this.totalTaskWaitingTime;
}

/** Get start time of the simulation */
public double getSimulationStartTime() {
		return simulationStartTime;
}

/**
 * Get total Task weight: LOOP all Task (numberOfCPU for execution * Task
 * actual CPU time)
 */
public double getTotalTaskWeight() {
		return totalTaskWeight;
}

/**
 * Get total slowdown of Tasks = Task response time / Task actual execution
 * time
 */
public double getTotalSlowdown() {
		return totalSlowdown;
}

/**
 * Get total weighted response time = (Task weight * Task response time) =
 * (Task used cpu number * Task actual cpu time * Task response time)
 */
public double getTotalWeightedResponseTime() {
		return totalWeightedResponseTime;
}

/**
 * Get total weighted slowdown = (Task weight * Task slowdown) = (Task used cpu
 * number * Task actual cpu time * Task slowdown)
 */
public double getTotalWeightedSlowdown() {
		return totalWeightedSlowdown;
```

```
    }

    /** Get number of Tasks submitted through submitter */
    public int getNumOfReceivedLocalTasks() {
                return numOfReceivedLocalTasks;
    }

    public double getAvgQueuingTime() {
                return avgQueuingTime;
    }

    public int sizeOfLocalTaskQueue() {
                return this.localTaskQueue.size();
    }

}
```

A.3. BGQoS Task info

```
package dsm.BGQoS.model;

import java.io.Serializable;
import java.util.Collections;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Map;
import java.util.Vector;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.concurrent.atomic.AtomicInteger;

import org.jfree.util.Log;

import dsm.BGQoS.BGQoSEntity;
import dsm.BGQoS.env.BGQoSMediator;
import dsm.BGQoS.env.BGQoSMessage;


/**
 * Class SimTaskInfo<p>
 * Task Owner(submitter/GRC) and MatchMaker Components.
 * It uses a set / get methods to set / get information about BGQoSTask.
 * It stores various information of the actual Task.
 * based on original GridSim Extention by @author Dalibor Klusacek */

public class TaskInfo implements Cloneable, Serializable {

   /** GRC id */
   private int userID;

   /** Task id */
   private int TaskLocalID;

   /** TaskInfo global id */
   private String globalTaskID;

   /** link to original Task */
   private Task Task;

   /** selected resource id */
   private int targetResourceID;

   /** computational length */
```

```
    private double computationalLength;
    private double TaskFinishedSoFar;
    private double cost;
    private double completitionFactor;

    /** reliability required by the Task */
    private float reliabilityRequired;

    /** memory required by the Task */
    private String memoryRequired;

    private float availabilityRequired;


    private double TaskStartTime;

    /** arrival time i.e. time of Task arrival in the system */
    private double arrivalTime;

   private double bandwidth;

    private double queuingStartTime;

    /** start time of Task processing */
    private double queuingEndTime;

    private double queuingTime;

    /** Time Constraint */
    private double deadline;


    /** It denotes this dynamicaly changing information: dynamicRealeaseTime = max(0.0, (arrivalTime +
TaskStartTime) - currentTime) */
    private double dynamicRealeaseTime;

    /** Task priority */
    private double TaskPriority;

    /** number of PEs to run this Task */
    private int numPE;

    /** estimated execution finish time */
    private double expectedFinishTime;

    /** estimated computational length */
    private double estimatedComputationTime;
```

```
/** MIPS rating of a machine used to compute estimated comp. length */
private double estimatedComputationalMIPS;

/** Task status */
private int TaskStatus;

/** id of original BGQoS where the Task is submitted */
private String originalBGQoSId = "";

/** id of BGQoS where the Task is executed */
private String executionBGQoSId = "";

/** Task profile for Partner execution */
private ConcurrentHashMap<String,Object> PartnerTaskInfoProfile   = null;

/** Task negotiation counter */
private AtomicInteger TaskNegotiationCounter = new AtomicInteger(0);

/**
 * Creates a new instance of TaskInfo object based on the BGQoSTask
 */
public TaskInfo(Task Task) {

   this.userID = Task.getUserID();
   this.setTaskLocalID(Task.getGridletID());
   this.setTaskStatus(Task.getTaskStatus());
   this.setComputationalLength(Task.getTaskLength());
   this.setTaskFinishedSoFar(Task.getTaskFinishedSoFar());
   this.setCompletitionFactor(Task.getTaskFinishedSoFar() / Task.getTaskLength());
   this.setTask(Task);
   this.setreliabilityRequired(Task.getReliabilityRequired());
   this.setmemoryRequired(Task.getMemoryRequired());
   this.setDeadline(Task.getDeadline());
   this.setTardiness(0.0);
   this.setDynamicRealeaseTime(0.0);
   this.setTaskPriority(Task.getTaskPriority());
   this.setNumPE(Task.getNumPE());
   this.setExpectedFinishTime(0);
   this.setEstimatedComputationTime(Task.getEstimatedComputationTime());
   this.setEstimatedComputationalMIPS(Task.getEstimatedComputationalMIPS());
   this.setArrivalTime(Task.getArrivalTime());
   this.setQueuingStartTime(Task.getArrivalTime());
   this.setQueuingEndTime(-1);
   this.setQueuingTime(BGQoSMediator.getSystemTime() - this.queuingStartTime);
   this.PartnerTaskInfoProfile = new ConcurrentHashMap<String,Object>();
   this.PartnerTaskInfoProfile.put(BGQoSMessage.MatchProfile_OS, Task.getOSRequired());

   this.PartnerTaskInfoProfile.put(BGQoSMessage.MatchProfile_CPUCount,                    new
```

```
Integer(Task.getNumPE()));
//      this.PartnerTaskInfoProfile.put(BGQoSMessage.MatchProfile_ExePrice, new Double(-2.0));


  }

  public int getUserID() {
    return userID;
  }


  public void setUserID(int userID) {
BGQoSTask Task = this.getTask();
Task.setUserID(userID);
    this.userID = userID;
    this.getTask().setUserID(userID);
  }


  public String getOriginalBGQoSId() {
            return originalBGQoSId;
  }

  public void setOriginalBGQoSId(String originalBGQoSId) {
            this.getTask().setOriginalBGQoSId(originalBGQoSId);
            this.globalTaskID = this.getTask().getGlobalTaskID();
            this.originalBGQoSId = originalBGQoSId;
  }

  public String getExecutionBGQoSId() {
            return executionBGQoSId;
  }

  public void setExecutionBGQoSId(String executionBGQoSId) {
            this.executionBGQoSId = executionBGQoSId;
  }

  public int getTaskLocalID() {
    return TaskLocalID;
  }

  public void setTaskLocalID(int TaskLocalID) {
    this.TaskLocalID = TaskLocalID;
  }

  public int getTargetResourceID() {
    return targetResourceID;
  }
```

```
public void setTargetResourceID(int targetResourceID) {
   this.targetResourceID = targetResourceID;
}

public int getTaskStatus() {
this.TaskStatus = getTask().getTaskStatus(); // essential for fresh information
   return TaskStatus;
}

public void setTaskStatus(int TaskStatus) {
   this.TaskStatus = TaskStatus;
}


public double getComputationalLength() {
   return computationalLength;
}

public void setComputationalLength(double computationalLength) {
   this.computationalLength = computationalLength;
}

public double getTaskFinishedSoFar() {
this.TaskFinishedSoFar = getTask().getTaskFinishedSoFar(); // essential for fresh information
   return TaskFinishedSoFar;
}

public void setTaskFinishedSoFar(double TaskFinishedSoFar) {
   this.TaskFinishedSoFar = TaskFinishedSoFar;

}

public double getCompletitionFactor() {
   return completitionFactor;
}

public void setCompletitionFactor(double completitionFactor) {
   this.completitionFactor = completitionFactor;
}


public String getReliabilityRequired() {
   return reliabilityRequired;
}

public void setReliabilityRequired(Float reliabilityRequired) {
   this.reliabilityRequired = reliabilityRequired;
```

```
}

public String getMemoryRequired() {
  return memoryRequired;
}

public void setMemoryRequired(String memoryRequired) {
  this.memoryRequired = memoryRequired;
}

public Task getTask() {
  return Task;
}

public void setTask(Task Task) {
  this.Task = Task;
}

public double getDeadline() {
  return deadline;
}
public void setDeadline(double deadline) {

  this.deadline = deadline;
}

public double getDynamicRealeaseTime() {
  return dynamicRealeaseTime;
}

public void setDynamicRealeaseTime(double dynamicRealeaseTime) {
  this.dynamicRealeaseTime = dynamicRealeaseTime;
}

public double getTaskPriority() {
  return TaskPriority;
}

public void setTaskPriority(double TaskPriority) {
  this.TaskPriority = TaskPriority;
}


public int getNumPE() {
  return numPE;
}

public void setNumPE(int numPE) {
```

```
    this.numPE = numPE;
  }

  public double getExpectedFinishTime() {
    return expectedFinishTime;
  }

  public void setExpectedFinishTime(double expectedFinishTime) {
    this.expectedFinishTime = expectedFinishTime;
  }

  public double getEstimatedComputationTime() {
    return estimatedComputationTime;
  }

  public void setEstimatedComputationTime(double estimatedComputationTime) {
    this.estimatedComputationTime = estimatedComputationTime;
  }

  public double getEstimatedComputationalMIPS() {
    return estimatedComputationalMIPS;
  }

  public void setEstimatedComputationalMIPS(double estimatedComputationalMIPS) {
    this.estimatedComputationalMIPS = estimatedComputationalMIPS;
  }

  /** Task negotiation counter */
  public AtomicInteger getTaskNegotiationCounter() {
            return TaskNegotiationCounter;
  }

  /** Task negotiation counter */
  public void setTaskNegotiationCounter(AtomicInteger TaskNegotiationCounter) {
            this.TaskNegotiationCounter = TaskNegotiationCounter;
  }

    public ConcurrentHashMap<String, Object> getPartnerTaskInfoProfile() {
            return PartnerTaskInfoProfile;
  }


  public void setPartnerTaskInfoProfile(
                    ConcurrentHashMap<String, Object> extPartnerTaskInfoProfile) {

            try {
this.PartnerTaskInfoProfile = new
ConcurrentHashMap <String,Object>();
```

```
 this.PartnerTaskInfoProfile.replace(BGQoSMessage.MatchProfile_OS,
extPartnerTaskInfoProfile.get(BGQoSMessage.MatchProfile_OS));
 this.PartnerTaskInfoProfile.replace(BGQoSMessage.MatchProfile_CPUCount,
extPartnerTaskInfoProfile.get(BGQoSMessage.MatchProfile_CPUCount));

                } catch(Exception e) {
                        e.printStackTrace();
                }

this.PartnerTaskInfoProfile = PartnerTaskInfoProfile;
  }

  /** update Task profile for Partner execution */
  public void updatePartnerTaskInfoProfile(String matchProfile_OS, Integer matchProfile_CPUCount) {

                this.PartnerTaskInfoProfile.replace(BGQoSMessage.MatchProfile_OS, matchProfile_OS);

    this.PartnerTaskInfoProfile.replace(BGQoSMessage.MatchProfile_CPUCount, matchProfile_CPUCount);

  }

  public String getGlobalTaskID() {
                return globalTaskID;
  }

  public double getArrivalTime() {
                // TODO Auto-generated method stub
                return this.arrivalTime;
  }

  public void setArrivalTime(double startTime) {
                // TODO Auto-generated method stub
                this.arrivalTime = startTime;
                this.getTask().setArrivalTime(startTime);
  }

  public double getQueuingStartTime()  {
                return this.queuingStartTime;
  }

  public void setQueuingStartTime(double queuingStartTime) {
                this.queuingStartTime = queuingStartTime;
  }

  public double getQueuingEndTime()  {
                return this.queuingEndTime;
  }
```

```
public void setQueuingEndTime(double queuingEndTime) {
        this.queuingEndTime = queuingEndTime;
}


public double getQueuingTime() {
        return this.queuingTime;
}

public void setQueuingTime(double queuingTime) {
        this.queuingTime = queuingTime;
}

public TaskInfo clone() {

        TaskInfo TaskInfo = null;
  try {
        TaskInfo = (TaskInfo) super.clone();
  } catch (Exception e) {
        e.printStackTrace();
  }
  return TaskInfo;
}
```

Appendix B

B.1. Introduction

The purpose for carrying out the simulations was to verify the functionality, feasibility, efficiency and practicality of BGQoS. While the results have been represented in tables, graphs and figures within this thesis, the result set provides a better sense of detail and specification. This appendix introduces a snippet of the generated results and the level of detail they produce.

B.2. Result Snippet

Total available MIPS power = 1152.0 MIPS in 1152.0 CPUs
>>> 10 so far arrived, in queue = 1 jobs, at time = 439959
>>> 20 so far arrived, in queue = 1 jobs, at time = 443219
>>> 30 so far arrived, in queue = 9 jobs, at time = 445455
>>> 40 so far arrived, in queue = 19 jobs, at time = 445869
*** 20 so far received, in queue = 16 jobs, at time = 446370
>>> 50 so far arrived, in queue = 11 jobs, at time = 446382
*** 30 so far received, in queue = 7 jobs, at time = 446400
*** 40 so far received, in queue = 2 jobs, at time = 446451
>>> 60 so far arrived, in queue = 1 jobs, at time = 464105
>>> 70 so far arrived, in queue = 1 jobs, at time = 487569
*** 70 so far received, in queue = 1 jobs, at time = 490803
>>> 80 so far arrived, in queue = 4 jobs, at time = 492157
>>> 90 so far arrived, in queue = 1 jobs, at time = 499755
>>> 100 so far arrived, in queue = 1 jobs, at time = 504940
>>> 110 so far arrived, in queue = 2 jobs, at time = 508127
>>> 120 so far arrived, in queue = 1 jobs, at time = 515173
>>> 130 so far arrived, in queue = 1 jobs, at time = 529008
*** 130 so far received, in queue = 3 jobs, at time = 540295
>>> 140 so far arrived, in queue = 1 jobs, at time = 555806
>>> 150 so far arrived, in queue = 1 jobs, at time = 563953
>>> 160 so far arrived, in queue = 1 jobs, at time = 565617
>>> 170 so far arrived, in queue = 1 jobs, at time = 566738
*** 170 so far received, in queue = 7 jobs, at time = 578772
>>> 180 so far arrived, in queue = 5 jobs, at time = 579313
>>> 190 so far arrived, in queue = 15 jobs, at time = 581246
>>> 200 so far arrived, in queue = 25 jobs, at time = 583087
>>> 210 so far arrived, in queue = 35 jobs, at time = 584497
*** 180 so far received, in queue = 20 jobs, at time = 584885
>>> 220 so far arrived, in queue = 1 jobs, at time = 588517
>>> 230 so far arrived, in queue = 1 jobs, at time = 590032

>>> 240 so far arrived, in queue = 1 jobs, at time = 591381
>>> 250 so far arrived, in queue = 1 jobs, at time = 592179
>>> 260 so far arrived, in queue = 1 jobs, at time = 593139
>>> 270 so far arrived, in queue = 1 jobs, at time = 595525
>>> 280 so far arrived, in queue = 1 jobs, at time = 596510
>>> 290 so far arrived, in queue = 1 jobs, at time = 597438
>>> 300 so far arrived, in queue = 1 jobs, at time = 597907
>>> 310 so far arrived, in queue = 1 jobs, at time = 599568
>>> 320 so far arrived, in queue = 1 jobs, at time = 599670
>>> 330 so far arrived, in queue = 1 jobs, at time = 600307
>>> 340 so far arrived, in queue = 1 jobs, at time = 601716
>>> 350 so far arrived, in queue = 1 jobs, at time = 602616
>>> 360 so far arrived, in queue = 1 jobs, at time = 603434
>>> 370 so far arrived, in queue = 1 jobs, at time = 603890
>>> 380 so far arrived, in queue = 1 jobs, at time = 605367
>>> 390 so far arrived, in queue = 1 jobs, at time = 606404
>>> 400 so far arrived, in queue = 1 jobs, at time = 608195
>>> 410 so far arrived, in queue = 1 jobs, at time = 610590
>>> 420 so far arrived, in queue = 1 jobs, at time = 612444
>>> 430 so far arrived, in queue = 1 jobs, at time = 615772
>>> 440 so far arrived, in queue = 1 jobs, at time = 617228
>>> 450 so far arrived, in queue = 1 jobs, at time = 619069
>>> 460 so far arrived, in queue = 1 jobs, at time = 620251
>>> 470 so far arrived, in queue = 1 jobs, at time = 621724
>>> 480 so far arrived, in queue = 1 jobs, at time = 622336
>>> 490 so far arrived, in queue = 1 jobs, at time = 622742
>>> 500 so far arrived, in queue = 1 jobs, at time = 623972
>>> 510 so far arrived, in queue = 1 jobs, at time = 627939
>>> 520 so far arrived, in queue = 1 jobs, at time = 630612
>>> 530 so far arrived, in queue = 1 jobs, at time = 631733
>>> 540 so far arrived, in queue = 1 jobs, at time = 635496
>>> 550 so far arrived, in queue = 1 jobs, at time = 642476
>>> 560 so far arrived, in queue = 1 jobs, at time = 645154
>>> 570 so far arrived, in queue = 1 jobs, at time = 653521
>>> 580 so far arrived, in queue = 1 jobs, at time = 663541
>>> 590 so far arrived, in queue = 2 jobs, at time = 664581
>>> 600 so far arrived, in queue = 8 jobs, at time = 668960
>>> 610 so far arrived, in queue = 18 jobs, at time = 671565
>>> 620 so far arrived, in queue = 25 jobs, at time = 674685
*** 590 so far received, in queue = 25 jobs, at time = 674947
*** 600 so far received, in queue = 12 jobs, at time = 675130
*** 610 so far received, in queue = 14 jobs, at time = 678057
>>> 630 so far arrived, in queue = 15 jobs, at time = 678938
>>> 640 so far arrived, in queue = 15 jobs, at time = 684765
*** 620 so far received, in queue = 12 jobs, at time = 685116
>>> 650 so far arrived, in queue = 1 jobs, at time = 687841
*** 640 so far received, in queue = 1 jobs, at time = 688167
>>> 660 so far arrived, in queue = 3 jobs, at time = 689568

>>> 670 so far arrived, in queue = 4 jobs, at time = 690117
>>> 680 so far arrived, in queue = 1 jobs, at time = 691515
>>> 690 so far arrived, in queue = 1 jobs, at time = 692549
*** 680 so far received, in queue = 1 jobs, at time = 692614
>>> 700 so far arrived, in queue = 1 jobs, at time = 693450
>>> 710 so far arrived, in queue = 1 jobs, at time = 695822
>>> 720 so far arrived, in queue = 1 jobs, at time = 697522
>>> 730 so far arrived, in queue = 1 jobs, at time = 699816
>>> 740 so far arrived, in queue = 1 jobs, at time = 699833
>>> 750 so far arrived, in queue = 1 jobs, at time = 701593
>>> 760 so far arrived, in queue = 1 jobs, at time = 705443
>>> 770 so far arrived, in queue = 1 jobs, at time = 705460
>>> 780 so far arrived, in queue = 1 jobs, at time = 707721
>>> 790 so far arrived, in queue = 1 jobs, at time = 710429
>>> 800 so far arrived, in queue = 1 jobs, at time = 712658
>>> 810 so far arrived, in queue = 1 jobs, at time = 718026
>>> 820 so far arrived, in queue = 1 jobs, at time = 722912
>>> 830 so far arrived, in queue = 1 jobs, at time = 724514
>>> 840 so far arrived, in queue = 1 jobs, at time = 724920
>>> 850 so far arrived, in queue = 1 jobs, at time = 725034
>>> 860 so far arrived, in queue = 1 jobs, at time = 725741
>>> 870 so far arrived, in queue = 1 jobs, at time = 725760
>>> 880 so far arrived, in queue = 1 jobs, at time = 725777
*** 10560 so far received, in queue = 5 jobs, at time = 4234422
>>> 10590 so far arrived, in queue = 6 jobs, at time = 4234433
*** 10570 so far received, in queue = 2 jobs, at time = 4234507
>>> 10600 so far arrived, in queue = 1 jobs, at time = 4235648
>>> 10610 so far arrived, in queue = 1 jobs, at time = 4236775
>>> 10620 so far arrived, in queue = 1 jobs, at time = 4238666
>>> 10630 so far arrived, in queue = 1 jobs, at time = 4239843
>>> 10640 so far arrived, in queue = 1 jobs, at time = 4241407
*** 10630 so far received, in queue = 2 jobs, at time = 4241852
>>> 10650 so far arrived, in queue = 1 jobs, at time = 4242784
>>> 10660 so far arrived, in queue = 1 jobs, at time = 4245436
>>> 10670 so far arrived, in queue = 1 jobs, at time = 4248005
>>> 10680 so far arrived, in queue = 1 jobs, at time = 4249992
Shuting down - last Gridlet = 12000 of 12000
End of submission... 10731

Machine usage = 66.52 % (used time/avail time) failures included. 0.0 % of failures.
Weighted machine usage = 66.51 % (used MIPS/avail MIPS) failures included. 0.0 % of failures.
---
0 = failed; Collected = Success + Failed : 10731 = 10731+0 | non-delayed = 2464
Total sched. time = 14642.0 ms | Makespan: 4357182.008
CHECK awsd: 8.612, Check slowdown=517.742619124589 -> 5555896.045825965/10731
Shuting down the blue_12000.swf_PWALoader... with: 1269 fails
Machine usage = 66.52 % 2898366.5523888227/4357164.1256 active/avail=0.0 / 1152.

Appendix C

C.1. Introduction

A portion of available information on the workflow of Grid'5000 has been used within the evaluation. Following is a snippet of how this information is structured.

C.2. Grid '5000 workflow

Generated by get-clean-log.py ($Revision: 0.1$) on Tue February 20, 2007, at 09:48:14 PM
Authors: Alexandru Iosup and Mathieu Jan ({A.Iosup|M.Jan} at tudelft.nl)
The Grid Workloads Archive (http://gwa.ewi.tudelft.nl/)
External coallocated_jobs info file: Grid5000_coallocated_jobs.log
External interactive_jobs info file: Grid5000_interactive_jobs.log
External reservation_jobs info file: Grid5000_reservation_jobs.log
External sites_time info file: Grid5000_sites_time.log
External user_to_group info file: Grid5000_user_to_group.log

Grid Workloads Format: JobId<TAB>SubmitTime<TAB>WaitTime<TAB>RunTime<TAB>NProc<TAB>AverageCPUTimeUsed<TAB>UsedMemory<TAB>ReqNProcs<TAB>ReqTime<TAB>ReqMemory<TAB>Status<TAB>UserId<TAB>GroupId<TAB>ExecutableId<TAB>QueueId<TAB>PartitionId<TAB>OrigSiteId<TAB>LastRunSiteId<TAB>JobStructure<TAB>JobStructureParams<TAB>UsedNetwork<TAB>UsedLocalDiskSpace<TAB>UsedResources<TAB>ReqPlatform<TAB>ReqNetwork<TAB>RequestedLocalDiskSpace<TAB>RequestedResources<TAB>VirtualOrganizationId<TAB>ProjectId

0	1083658801		1	0	4	-1	-1	4	3600	-1											
	1	user386	group4	app34	queue0	-1	G1/site4	G1/site6/c1		UNITARY											
	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1											
1	1083658849		1	19	1	-1	-1	1	3600	-1											
	1	user112	group6	app0	queue0	-1	G1/site6	G1/site6/c1		UNITARY											
	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1											
2	1083658875		2	10	5	-1	-1	5	3600	-1											
	1	user112	group6	app0	queue0	-1	G1/site6	G1/site6/c1		UNITARY											
	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1											
3	1083658891		5	8	90	-1	-1	90	3600	-1											
	1	user112	group6	app0	queue0	-1	G1/site6	G1/site6/c1		UNITARY											
	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1											
4	1083658911		5	19	100	-1	-1	100	3600	-1											
	1	user112	group6	app0	queue0	-1	G1/site6	G1/site6/c1		UNITARY											
	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1											
5	1083658944		1	25	1	-1	-1	1	3600	-1											
	0	user112	group6	app0	queue0	-1	G1/site6	G1/site6/c1		UNITARY											
	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1											
6	1083659210		1	6	1	-1	-1	1	3600	-1											

```
  1         user568   group6    app0      queue0    -1        G1/site6  G1/site6/c1       UNITARY
  -1        -1        -1        -1        -1        -1        -1        -1        -1        -1
7 1083659322        1         43205     4         -1        -1        4         43200     -1
  0         user386   group4    app0      queue0    -1        G1/site4  G1/site6/c1       UNITARY
  -1        -1        -1        -1        -1        -1        -1        -1        -1        -1
8 1083659636        1         5         1         -1        -1        1         3600      -1
  1         user568   group6    app0      queue0    -1        G1/site6  G1/site6/c1       UNITARY
  -1        -1        -1        -1        -1        -1        -1        -1        -1        -1
9 1083660389        -1        -1        4         -1        -1        4         9000      -1
  0         user267   group5    app507    queue48   -1        G1/site5  G1/site6/c1       UNITARY
  -1        -1        -1        -1        -1        -1        -1        -1        -1        -1
10        1083660523        2         156       7         -1        -1        7         18000
  -1        1         user569   group6    app0      queue0    -1        G1/site6  G1/site6/c1
  UNITARY -1        -1        -1        -1        -1        -1        -1        -1        -1
  -1
11        1083660693        1         19        7         -1        -1        7         18000
  -1        1         user569   group6    app0      queue0    -1        G1/site6  G1/site6/c1
  UNITARY -1        -1        -1        -1        -1        -1        -1        -1        -1
  -1
12        1083660719        1         4         7         -1        -1        7         18000
  -1        1         user569   group6    app0      queue0    -1        G1/site6  G1/site6/c1
  UNITARY -1        -1        -1        -1        -1        -1        -1        -1        -1
  -1
13        1083660726        1         2801      7         -1        -1        7         18000
  -1        1         user569   group6    app0      queue0    -1        G1/site6  G1/site6/c1
  UNITARY -1        -1        -1        -1        -1        -1        -1        -1        -1
  -1
14        1083660777        1         1         4         -1        -1        4         3600
  -1        1         user267   group5    app507    queue0    -1        G1/site5  G1/site6/c1
  UNITARY -1        -1        -1        -1        -1        -1        -1        -1        -1
  -1
15        1083660832        1         0         4         -1        -1        4         3600
  -1        1         user267   group5    app507    queue0    -1        G1/site5  G1/site6/c1
  UNITARY -1        -1        -1        -1        -1        -1        -1        -1        -1
  -1
16        1083660933        1         0         4         -1        -1        4         3600
  -1        1         user267   group5    app507    queue0    -1        G1/site5  G1/site6/c1
  UNITARY -1        -1        -1        -1        -1        -1        -1        -1        -1
  -1
17        1083661197        1         20992     1         -1        -1        1         36000
  -1        1         user570   group6    app0      queue0    -1        G1/site6  G1/site6/c1
  UNITARY -1        -1        -1        -1        -1        -1        -1        -1        -1
  -1
18        1083661769        1         23027     1         -1        -1        1         43200
  -1        1         user571   group6    app0      queue0    -1        G1/site6  G1/site6/c1
  UNITARY -1        -1        -1        -1        -1        -1        -1        -1        -1
  -1
19        1083661777        1         23014     1         -1        -1        1         43200
```

```
   -1        1        user571  group6   app0     queue0   -1       G1/site6  G1/site6/c1
   UNITARY -1         -1       -1       -1       -1       -1       -1        -1        -1
   -1
20         1083662072       1        22216    1        -1       -1        1         28800
   -1        1        user67   group2   app0     queue0   -1       G1/site2  G1/site6/c1
   UNITARY -1         -1       -1       -1       -1       -1       -1        -1        -1
   -1
21         1083663533       1        13734    10       -1       -1        10        18000
   -1        0        user569  group6   app0     queue0   -1       G1/site6  G1/site6/c1
   UNITARY -1         -1       -1       -1       -1       -1       -1        -1
```

Appendix D BGQoS Database Structure

D.1. Introduction

This appendix shows aspects of the database implementation. The database implementation served as an access point for up-to-date information , as reference when reallocation occurs, agreement referencing and storing information on GRCs, GRPs and Broker.

D.2. Some Relations and Tables

Server: localhost ▸ Database: mydb

agreement

Field	Type	Null	Default	Comments
id	int(11)	No		
name	varchar(45)	Yes	*NULL*	
template_id	int(11)	Yes	*NULL*	
application_id	int(11)	Yes	*NULL*	
grc_id	int(11)	Yes	*NULL*	
grp_id	int(11)	Yes	*NULL*	

application

Field	Type	Null	Default	Comments
id	int(11)	No		
grc_id	int(11)	Yes	*NULL*	

broker

Field	Type	Null	Default	Comments
id	int(11)	No		
address	varchar(45)	Yes	*NULL*	

grc

Field	Type	Null	Default	Comments
id	int(11)	No		
name	varchar(45)	Yes	*NULL*	
tier	int(11)	Yes	*NULL*	

grc_tier

Field	Type	Null	Default	Comments
tier	int(11)	No		
description	varchar(45)	Yes	*NULL*	

grp

Field	Type	Null	Default	Comments
id	int(11)	No		
name	varchar(45)	Yes	*NULL*	
address	varchar(45)	Yes	*NULL*	

permission

Field	Type	Null	Default	Comments
id	int(11)	No		
description	varchar(45)	Yes	*NULL*	

policy

Field	Type	Null	Default	Comments
id	int(11)	No		
resource_id	int(11)	Yes	*NULL*	
grp_id	int(11)	Yes	*NULL*	

resource

Field	Type	Null	Default	Comments
id	int(11)	No		
type	varchar(45)	Yes	*NULL*	
domain	varchar(45)	Yes	*NULL*	
cpu_count	double	Yes	*NULL*	
cpu	double	Yes	*NULL*	
availability	double	Yes	*NULL*	
reliability	double	Yes	*NULL*	
memory	double	Yes	*NULL*	
bandwidth	double	Yes	*NULL*	
storage	double	Yes	*NULL*	
grp_id	int(11)	Yes	*NULL*	

tier_permission

Field	Type	Null	Default	Comments
tier	int(11)	No		
permission_id	int(11)	No		

D.3. XML Schema Snippet

```
- <database name="test1 ">
- <table_structure name="agreement">
  <field Field="id" Type="int(11)" Null="NO" Key="PRI" Extra="" />
  <field Field="name" Type="varchar(45)" Null="YES" Key="" Extra="" />
  <field Field="template_id" Type="int(11)" Null="YES" Key="" Extra="" />
  <field Field="application_id" Type="int(11)" Null="YES" Key="" Extra="" />
  <field Field="grc_id" Type="int(11)" Null="YES" Key="" Extra="" />
  <field Field="grp_id" Type="int(11)" Null="YES" Key="" Extra="" />
  <key Table="agreement" Non_unique="0" Key_name="PRIMARY" Seq_in_index="1" Column_name="id"
       Collation="A" Cardinality="0" Null="" Index_type="BTREE" Comment="" Index_comment="" />
  <options Name="agreement" Engine="InnoDB" Version="10" Row_format="Compact" Rows="0"
       Avg_row_length="0" Data_length="16384" Max_data_length="0" Index_length="0" Data_free="4194304"
       Create_time="2011-06-16 12:26:22" Collation="utf8_general_ci" Create_options="" Comment="" />
    </table_structure>
  <table_data name="agreement" />
- <table_structure name="application">
  <field Field="id" Type="int(11)" Null="NO" Key="PRI" Extra="" />
  <field Field="grc_id" Type="int(11)" Null="YES" Key="" Extra="" />
  <key Table="application" Non_unique="0" Key_name="PRIMARY" Seq_in_index="1" Column_name="id"
       Collation="A" Cardinality="0" Null="" Index_type="BTREE" Comment="" Index_comment="" />
  <options Name="application" Engine="InnoDB" Version="10" Row_format="Compact" Rows="0"
       Avg_row_length="0" Data_length="16384" Max_data_length="0" Index_length="0" Data_free="4194304"
       Create_time="2011-06-16 12:27:01" Collation="utf8_general_ci" Create_options="" Comment="" />
    </table_structure>
  <table_data name="application" />
- <table_structure name="broker">
  <field Field="id" Type="int(11)" Null="NO" Key="PRI" Extra="" />
  <field Field="address" Type="varchar(45)" Null="YES" Key="" Extra="" />
  <key Table="broker" Non_unique="0" Key_name="PRIMARY" Seq_in_index="1" Column_name="id"
       Collation="A" Cardinality="0" Null="" Index_type="BTREE" Comment="" Index_comment="" />
  <options Name="broker" Engine="InnoDB" Version="10" Row_format="Compact" Rows="0"
       Avg_row_length="0" Data_length="16384" Max_data_length="0" Index_length="0" Data_free="4194304"
       Create_time="2011-06-16 12:27:37" Collation="utf8_general_ci" Create_options="" Comment="" />
    </table_structure>
  <table_data name="broker" />
- <table_structure name="grc">
  <field Field="id" Type="int(11)" Null="NO" Key="PRI" Extra="" />
  <field Field="name" Type="varchar(45)" Null="YES" Key="" Extra="" />
  <field Field="tier" Type="int(11)" Null="YES" Key="" Extra="" />
  <key Table="grc" Non_unique="0" Key_name="PRIMARY" Seq_in_index="1" Column_name="id" Collation="A"
       Cardinality="0" Null="" Index_type="BTREE" Comment="" Index_comment="" />
  <options Name="grc" Engine="InnoDB" Version="10" Row_format="Compact" Rows="0" Avg_row_length="0"
       Data_length="16384" Max_data_length="0" Index_length="0" Data_free="4194304" Create_time="2011-
       06-16 12:07:40" Collation="utf8_general_ci" Create_options="" Comment="" />
    </table_structure>
  <table_data name="grc" />
- <table_structure name="grc_tier">
  <field Field="tier" Type="int(11)" Null="NO" Key="PRI" Extra="" />
  <field Field="description" Type="varchar(45)" Null="YES" Key="" Extra="" />
  <key Table="grc_tier" Non_unique="0" Key_name="PRIMARY" Seq_in_index="1" Column_name="tier"
       Collation="A" Cardinality="0" Null="" Index_type="BTREE" Comment="" Index_comment="" />
  <options Name="grc_tier" Engine="InnoDB" Version="10" Row_format="Compact" Rows="0"
       Avg_row_length="0" Data_length="16384" Max_data_length="0" Index_length="0" Data_free="4194304"
       Create_time="2011-06-16 12:10:55" Collation="utf8_general_ci" Create_options="" Comment="" />
    </table_structure>
  <table_data name="grc_tier" />
- <table_structure name="grp">
  <field Field="id" Type="int(11)" Null="NO" Key="PRI" Extra="" />
  <field Field="name" Type="varchar(45)" Null="YES" Key="" Extra="" />
  <field Field="address" Type="varchar(45)" Null="YES" Key="" Extra="" />
  <key Table="grp" Non_unique="0" Key_name="PRIMARY" Seq_in_index="1" Column_name="id" Collation="A"
       Cardinality="0" Null="" Index_type="BTREE" Comment="" Index_comment="" />
  <options Name="grp" Engine="InnoDB" Version="10" Row_format="Compact" Rows="0" Avg_row_length="0"
       Data_length="16384" Max_data_length="0" Index_length="0" Data_free="4194304" Create_time="2011-
       06-16 12:13:12" Collation="utf8_general_ci" Create_options="" Comment="" />
    </table_structure>
  <table_data name="grp" />
- <table_structure name="permission">
```

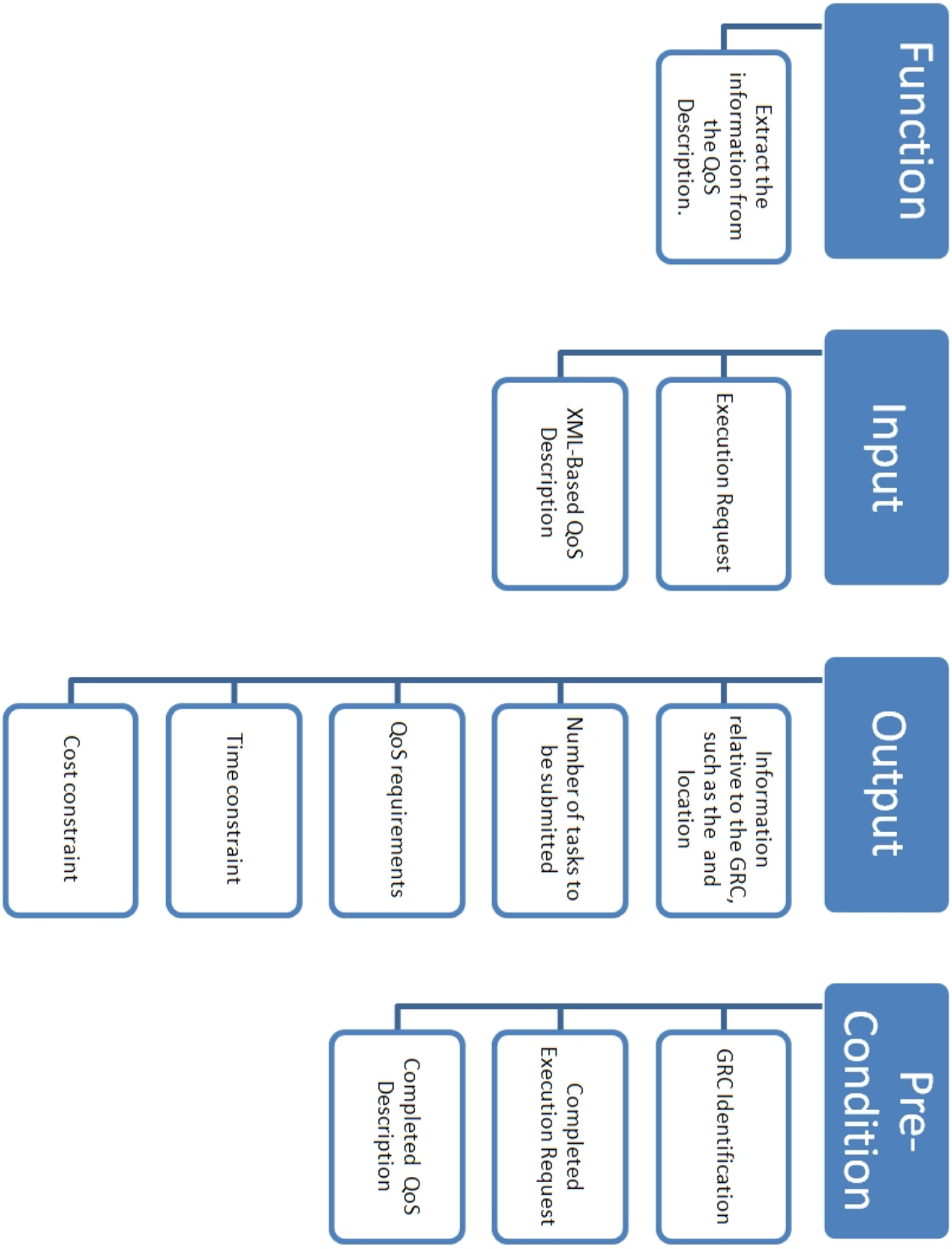
```
<field Field="id" Type="int(11)" Null="NO" Key="PRI" Extra="" />
<field Field="description" Type="varchar(45)" Null="YES" Key="" Extra="" />
<key Table="permission" Non_unique="0" Key_name="PRIMARY" Seq_in_index="1" Column_name="id"
    Collation="A" Cardinality="0" Null="" Index_type="BTREE" Comment="" Index_comment="" />
<options Name="permission" Engine="InnoDB" Version="10" Row_format="Compact" Rows="0"
    Avg_row_length="0" Data_length="16384" Max_data_length="0" Index_length="0" Data_free="4194304"
    Create_time="2011-06-16 12:12:34" Collation="utf8_general_ci" Create_options="" Comment="" />
</table_structure>
<table_data name="permission" />
- <table_structure name="policy">
<field Field="id" Type="int(11)" Null="NO" Key="PRI" Extra="" />
<field Field="resource_id" Type="int(11)" Null="YES" Key="" Extra="" />
<field Field="grp_id" Type="int(11)" Null="YES" Key="" Extra="" />
<key Table="policy" Non_unique="0" Key_name="PRIMARY" Seq_in_index="1" Column_name="id"
    Collation="A" Cardinality="0" Null="" Index_type="BTREE" Comment="" Index_comment="" />
<options Name="policy" Engine="InnoDB" Version="10" Row_format="Compact" Rows="0"
    Avg_row_length="0" Data_length="16384" Max_data_length="0" Index_length="0" Data_free="4194304"
    Create_time="2011-06-17 15:44:28" Collation="utf8_general_ci" Create_options="" Comment="" />
</table_structure>
<table_data name="policy" />
- <table_structure name="resource">
<field Field="id" Type="int(11)" Null="NO" Key="PRI" Extra="" />
<field Field="type" Type="varchar(45)" Null="YES" Key="" Extra="" />
<field Field="domain" Type="varchar(45)" Null="YES" Key="" Extra="" />
<field Field="cpu_count" Type="double" Null="YES" Key="" Extra="" />
<field Field="cpu" Type="double" Null="YES" Key="" Extra="" />
<field Field="availability" Type="double" Null="YES" Key="" Extra="" />
<field Field="reliability" Type="double" Null="YES" Key="" Extra="" />
<field Field="memory" Type="double" Null="YES" Key="" Extra="" />
<field Field="bandwidth" Type="double" Null="YES" Key="" Extra="" />
<field Field="storage" Type="double" Null="YES" Key="" Extra="" />
<field Field="grp_id" Type="int(11)" Null="YES" Key="" Extra="" />
<key Table="resource" Non_unique="0" Key_name="PRIMARY" Seq_in_index="1" Column_name="id"
    Collation="A" Cardinality="0" Null="" Index_type="BTREE" Comment="" Index_comment="" />
<options Name="resource" Engine="InnoDB" Version="10" Row_format="Compact" Rows="0"
    Avg_row_length="0" Data_length="16384" Max_data_length="0" Index_length="0" Data_free="4194304"
    Create_time="2011-06-16 12:14:22" Collation="utf8_general_ci" Create_options="" Comment="" />
</table_structure>
<table_data name="resource" />
- <table_structure name="tier_permission">
<field Field="tier" Type="int(11)" Null="NO" Key="PRI" Extra="" />
<field Field="permission_id" Type="int(11)" Null="NO" Key="PRI" Extra="" />
<key Table="tier_permission" Non_unique="0" Key_name="PRIMARY" Seq_in_index="1" Column_name="tier"
    Collation="A" Cardinality="0" Null="" Index_type="BTREE" Comment="" Index_comment="" />
<key Table="tier_permission" Non_unique="0" Key_name="PRIMARY" Seq_in_index="2"
    Column_name="permission_id" Collation="A" Cardinality="0" Null="" Index_type="BTREE" Comment=""
    Index_comment="" />
<options Name="tier_permission" Engine="InnoDB" Version="10" Row_format="Compact" Rows="0"
    Avg_row_length="0" Data_length="16384" Max_data_length="0" Index_length="0" Data_free="4194304"
    Create_time="2011-06-16 12:12:00" Collation="utf8_general_ci" Create_options="" Comment="" />
</table_structure>
<table_data name="tier_permission" />
</database>
```

Appendix E

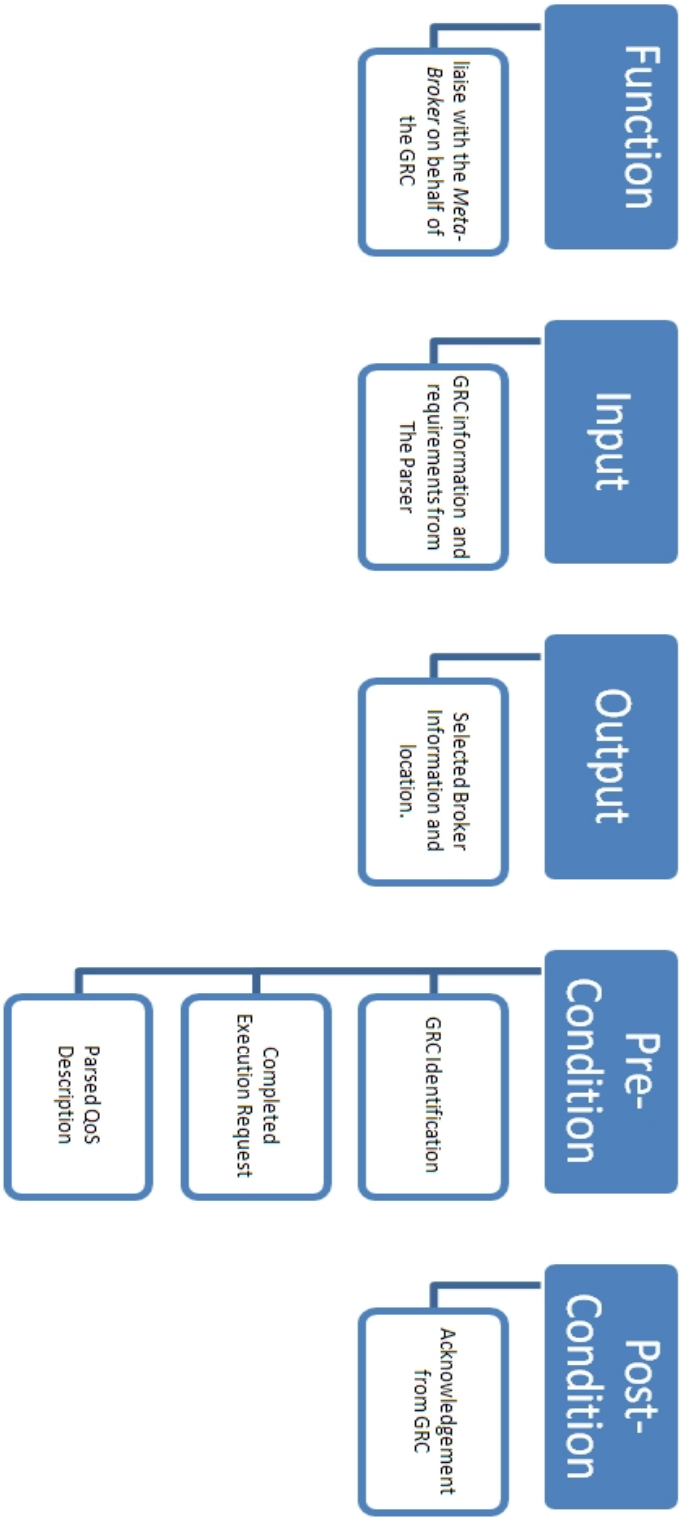
E.1. Introduction

This appendix includes a description of the functions, inputs, outputs and conditions for the major components of BGQoS explained in Chapter 5.

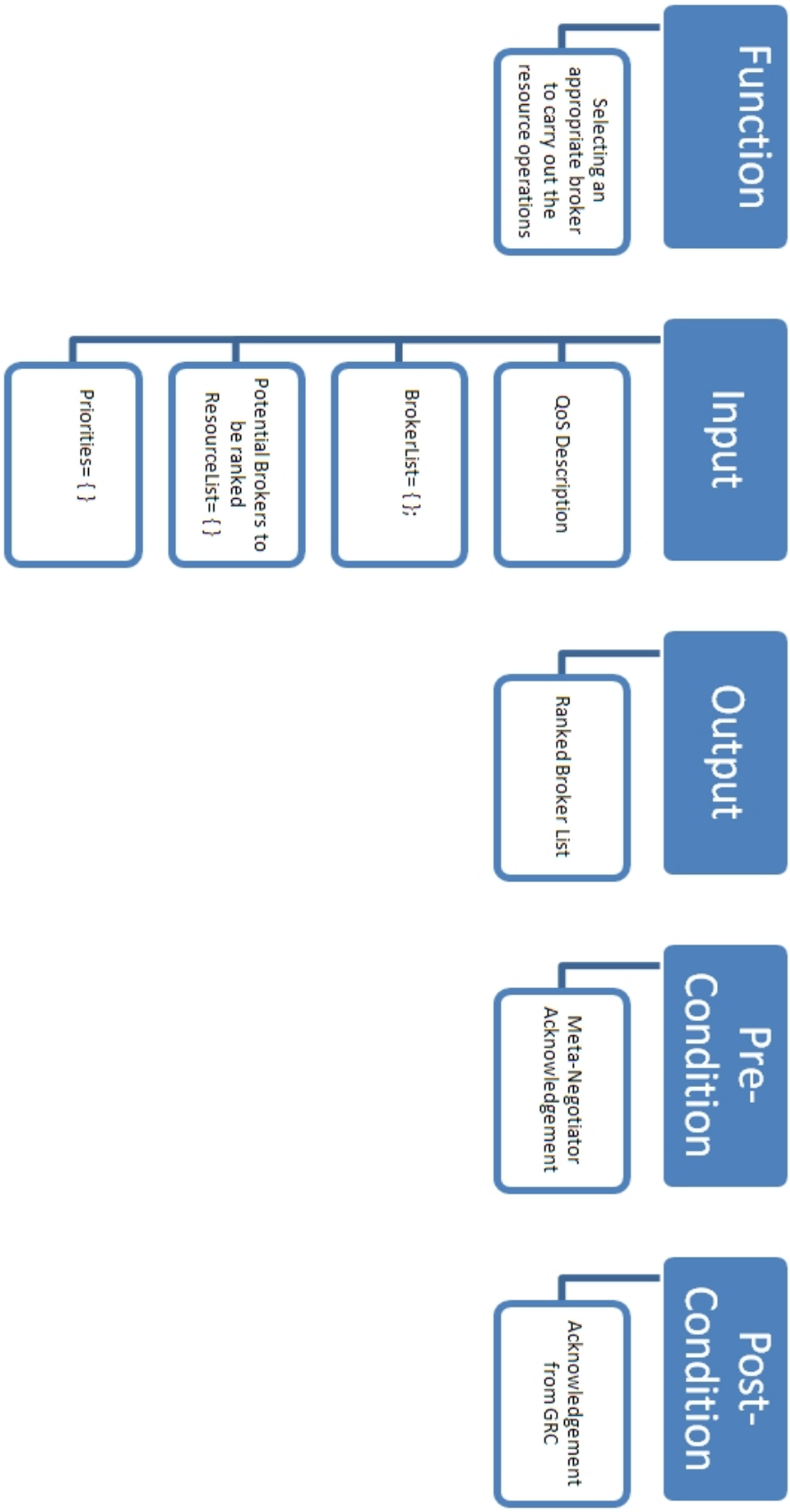
E.2. $QoS_{description}$ parser



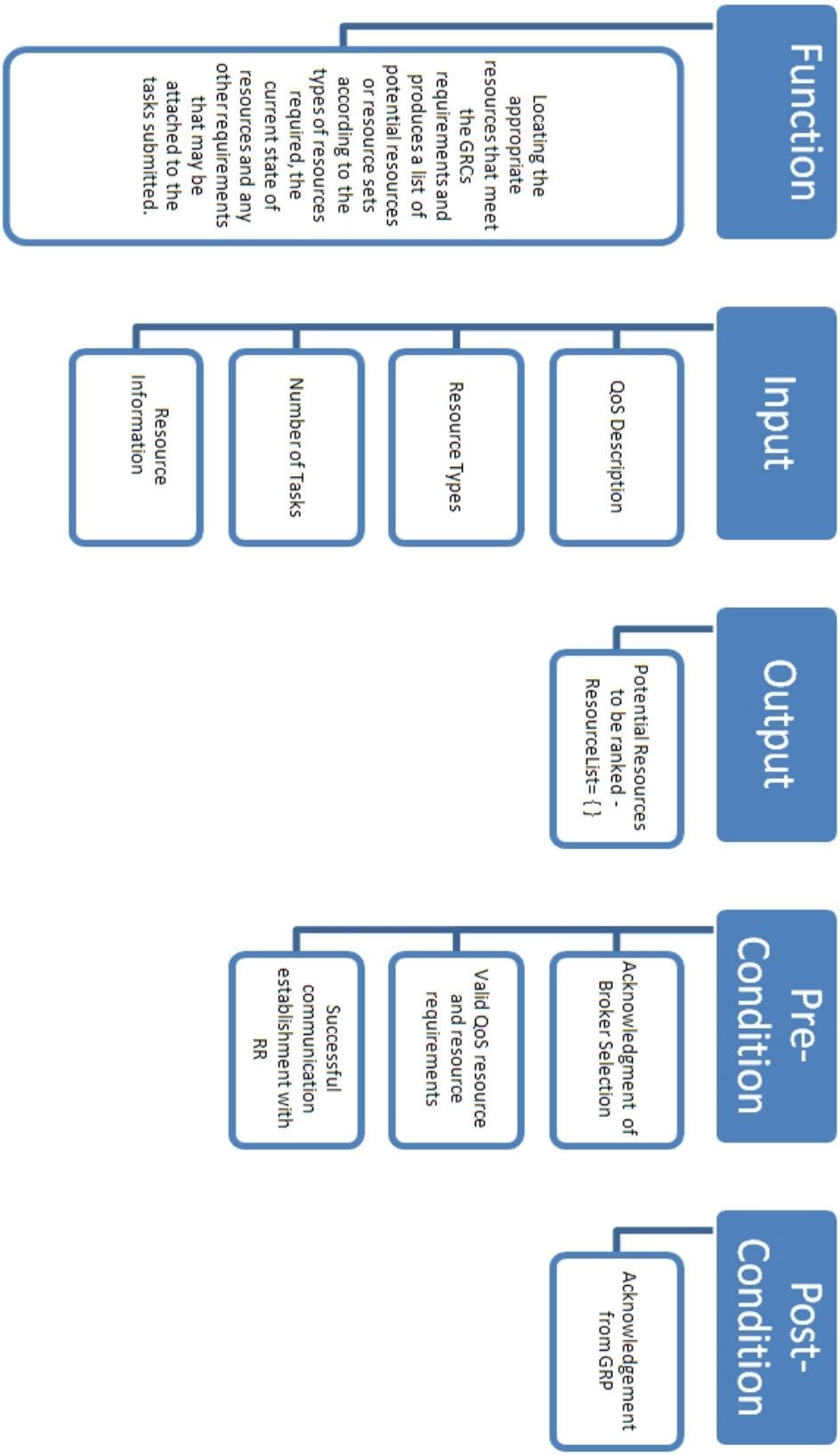
E.3. Meta-Negotiator



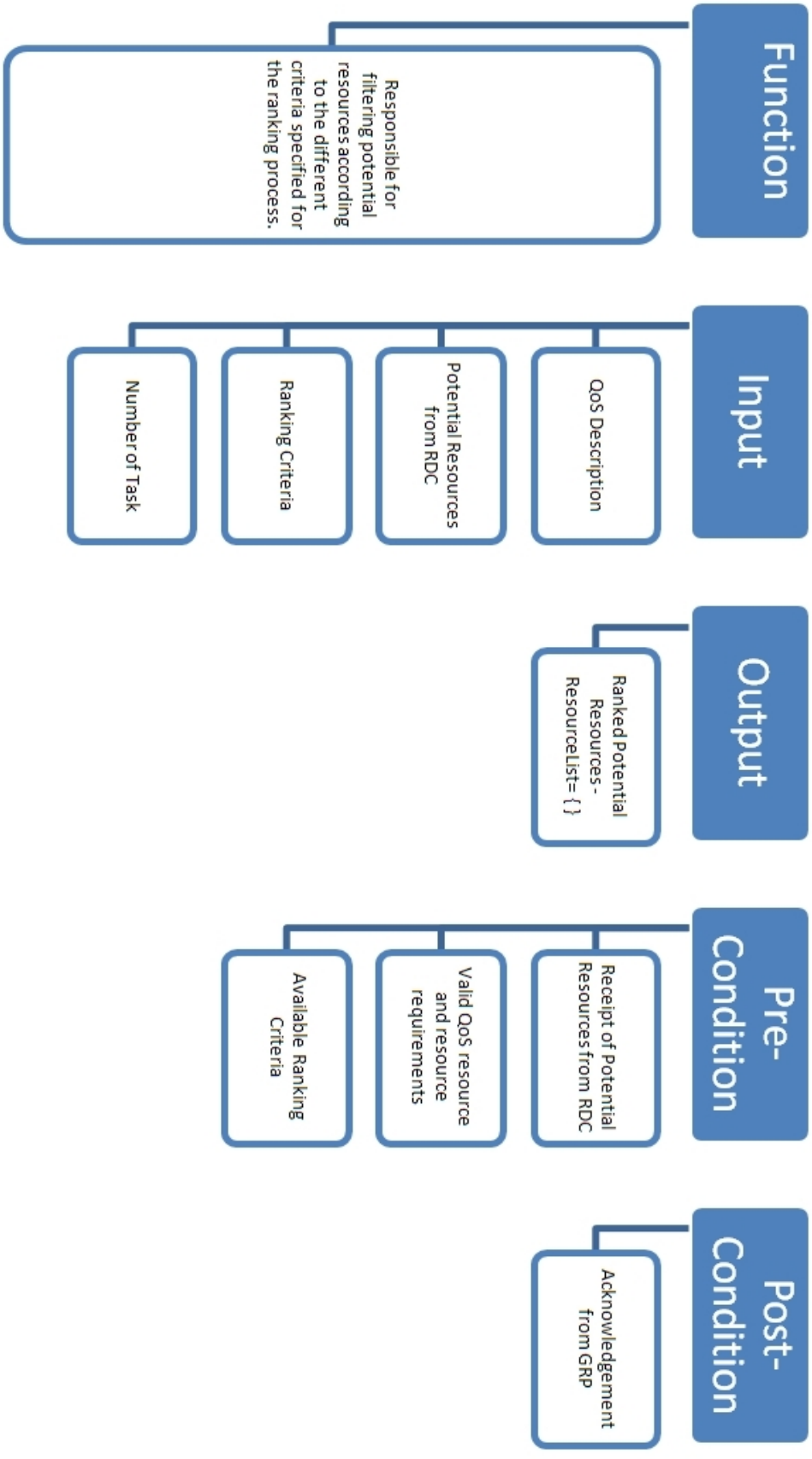
E.4. Meta-Broker



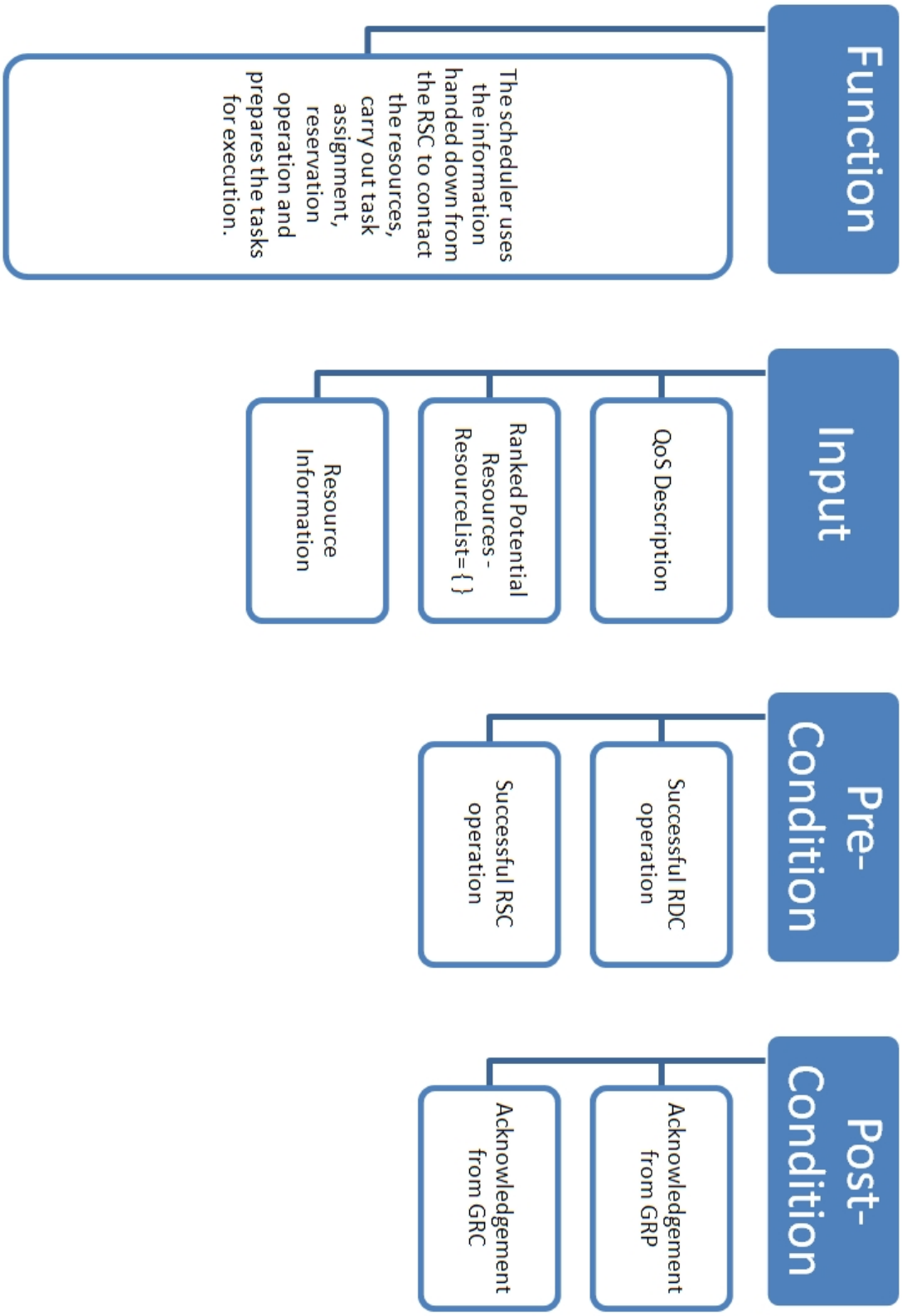
E.5. Resource Discovery Component



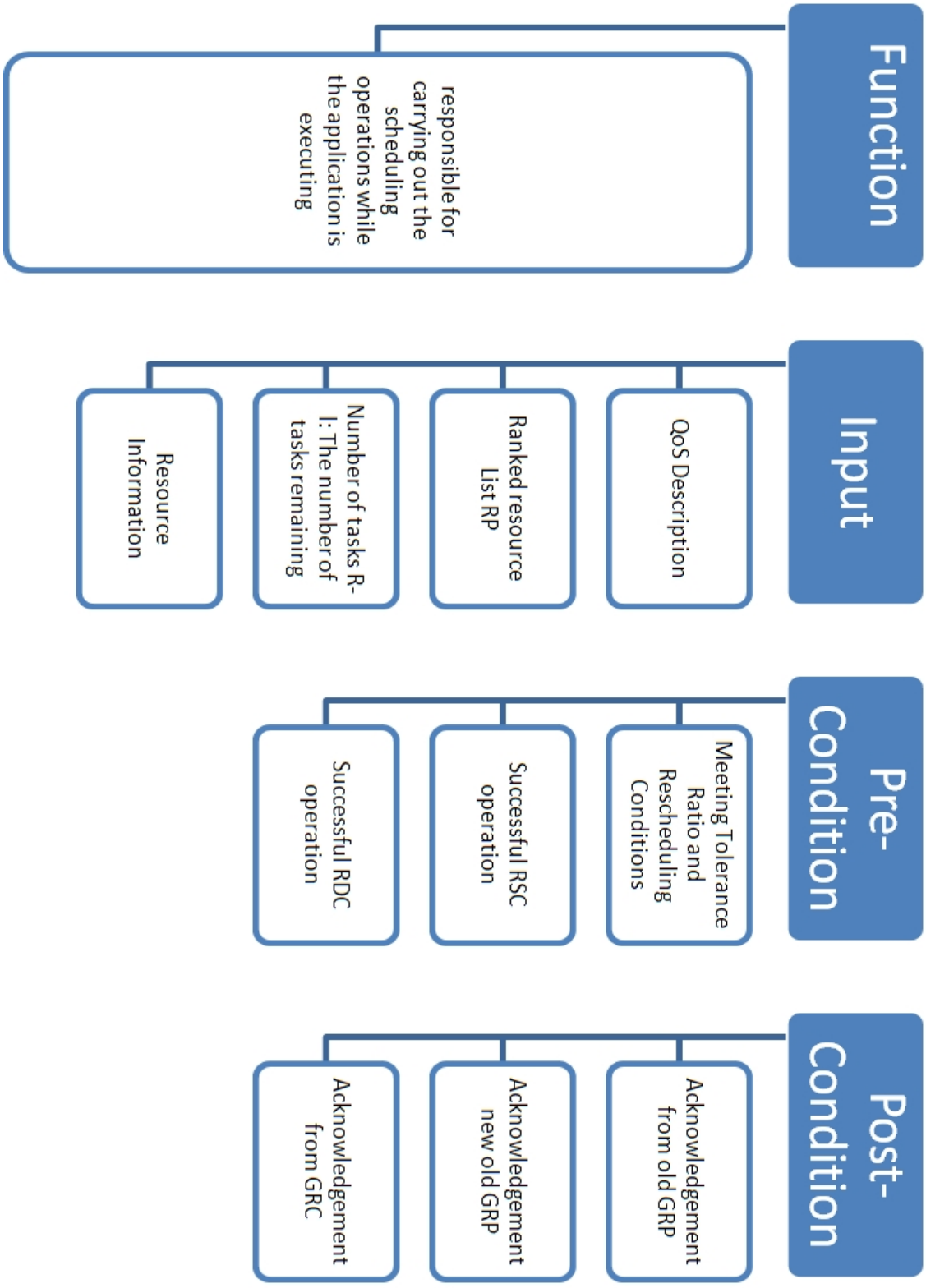
E.6. Resource Selection Component



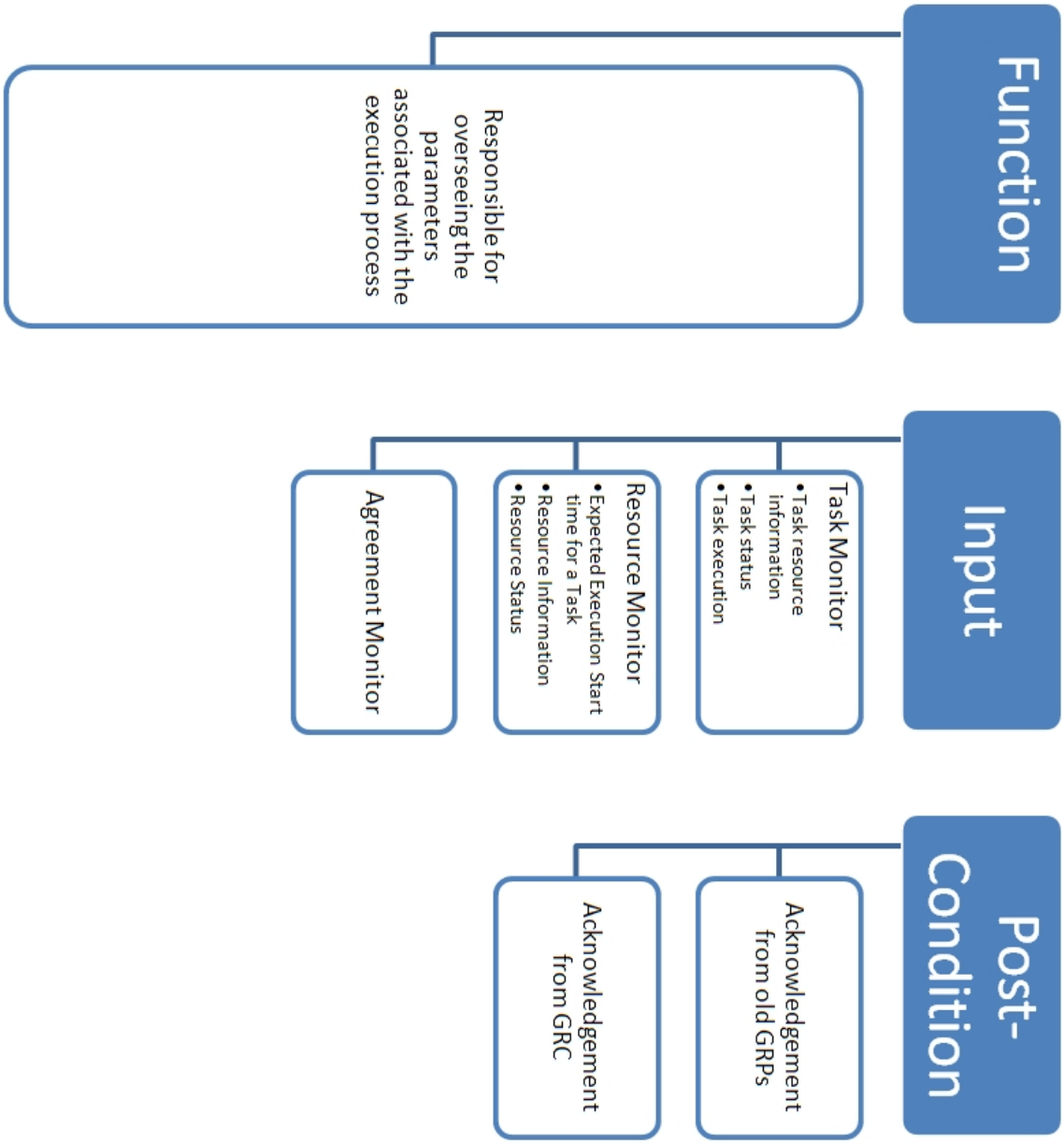
E.7. Scheduling Component



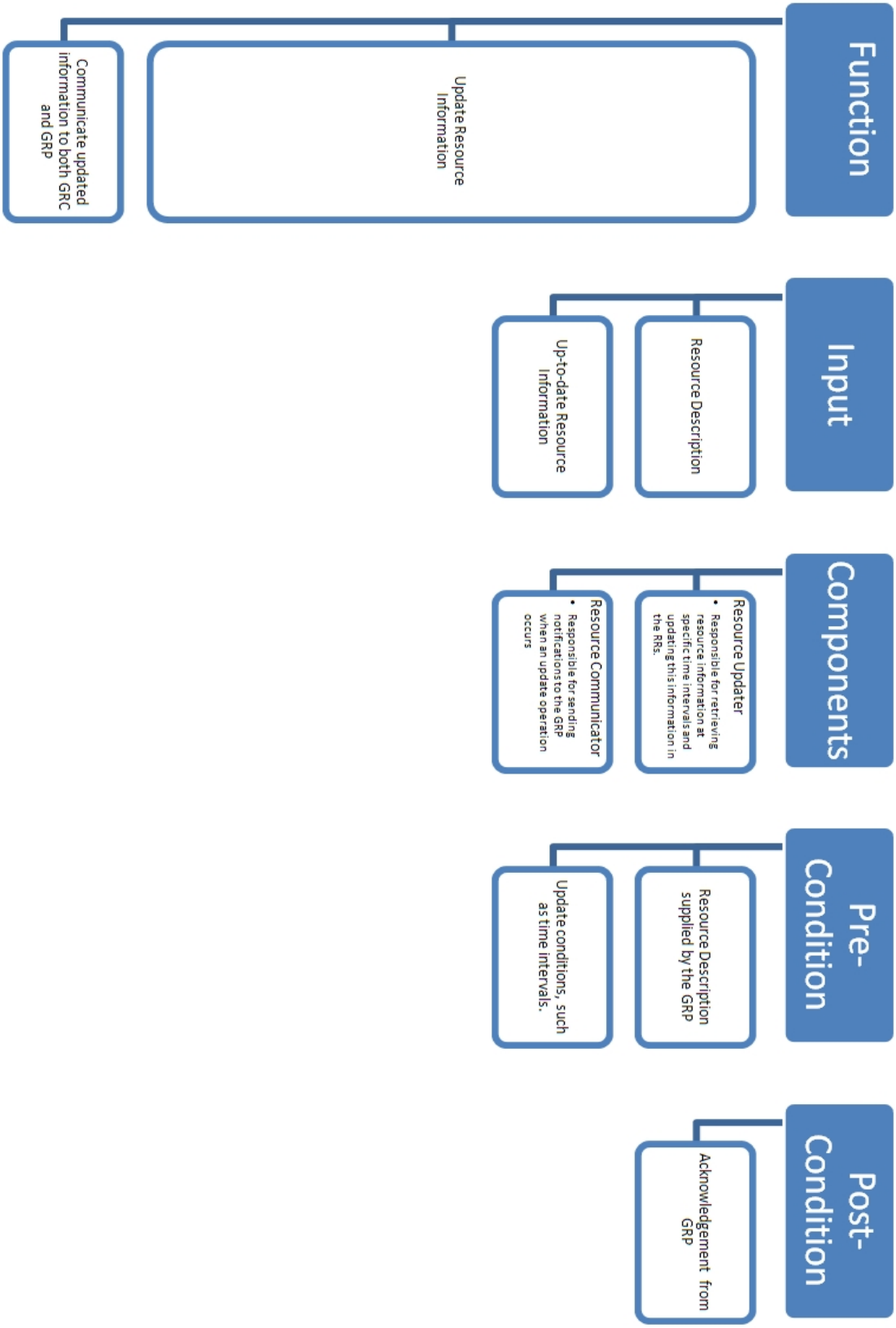
E.8. Rescheduler Component



E.9. Monitoring Component



E.10. Resource Management Component



E.11. Task Launcher